

USER GUIDE

Essential Studio

for EJ2 Angular

Version - v25.2.3 | Release Date - May 08, 2024

Pivot Table	32
Getting started with Angular Pivotview component	32
Setup Angular Environment.....	32
Create an Angular Application	32
Dependencies.....	32
Installing Syncfusion PivotView package	33
Registering PivotView Module	34
Adding CSS reference.....	34
Browser compatibility	34
Initializing pivot table component in an application	35
Assigning sample data to the pivot table.....	35
Adding fields to row, column, value and filter axes.....	36
Applying formatting to a value field	38
Module Injection.....	39
Enable Field List.....	39
Enable Grouping Bar	40
Exploring Filter Axis.....	41
Calculated field	42
Run the application	44
Data binding in Angular Pivotview component	45
JSON	45
CSV	50
Remote Data Binding	53
Mapping	58
Values in row axis.....	60
Values at different positions	61
Show 'no data' items.....	62
Show value headers always	63
Customize empty value cells.....	64
Event	65
See Also	68
Connecting to data source	68
MySQL in EJ2 Angular Pivotview Component.....	68
Microsoft SQL Server in EJ2 Angular Pivotview Component	74
PostgreSQL in EJ2 Angular Pivotview Component.....	79

Oracle in EJ2 Angular Pivotview Component.....	84
MongoDB in EJ2 Angular Pivotview Component	90
Elasticsearch in EJ2 Angular Pivotview Component	96
Snowflake in EJ2 Angular Pivotview Component.....	101
Olap in Angular Pivot Table component	107
Getting started	107
Data Binding.....	137
OLAP Cube: Elements.....	145
Server side pivot engine in Angular Pivotview component	147
Quick steps to render the Pivot Table by using the server-side Pivot Engine	148
Available configurations in Server-side application	150
Pivot chart in Angular Pivotview component	167
Data Binding.....	169
Chart Types	169
Accumulation Charts.....	170
Field List	181
Grouping Bar	182
Single Axis	184
Multiple Axis	185
Series customization	191
Axis Customization	194
Legend customization	196
User interaction	198
Export.....	202
Print.....	204
Events.....	205
Drill down in Angular Pivotview component	207
Drill down and drill up.....	207
Drill position	207
Expand All.....	208
Expand all headers for specific fields.....	209
Expand all except specific member(s).....	210
Expand specific member(s)	211
Event	212
Data Shaping	217

Aggregation in Angular Pivotview component	217
Calculated field in Angular Pivotview component	229
Grouping in Angular Pivotview component	239
Filtering in Angular Pivotview component	239
Sorting in Angular Pivotview component	240
Drill through in Angular Pivotview component	240
Editing in Angular Pivotview component	240
Data Formatting	240
Number formatting in Angular Pivotview component	240
Conditional formatting in Angular Pivotview component	240
Report Manipulation	240
Grouping bar in Angular Pivotview component	240
Field list in Angular Pivotview component	240
Defer update in Angular Pivotview component	273
Performance	276
Virtual scrolling in Angular Pivot Table component	276
Paging in Angular Pivotview component	282
Data Compression in Angular Pivot Table component	295
State persistence in Angular Pivotview component	297
Save and Load Pivot Layout	298
Row and column in Angular Pivotview component	300
Width and Height	300
Row Height	301
Column Width	302
Reorder	304
Column Resizing	305
Text Wrap	307
Text Align	308
AutoFit	309
Grid Lines	313
Selection	314
Clip Mode	322
Cell Template	323
Events	326
Summary customization in Angular Pivotview component	331

Show or hide grand totals	331
Show Grand Totals at Top	332
Show or hide sub-totals	333
Show or hide sub-totals for specific fields	335
Show sub-totals at top or bottom.....	336
Show or hide totals using toolbar	338
Hyper link in Angular Pivotview component	339
Hyperlink for all cells.....	339
Hyperlink for row headers	340
Hyperlink for column headers.....	341
Hyperlink for value cells.....	342
Hyperlink for summary cells	344
Condition based hyperlink	345
Header based hyperlink	346
Event	347
See Also	348
Tool bar in Angular Pivotview component	348
Show desired chart types in the dropdown menu	352
Switch the chart to multiple axes	353
Show or hide legend	354
Adding custom option to the toolbar	355
Save and load report as a JSON file.....	360
Save and load reports to a SQL database	362
Events.....	387
See Also	399
Tool tip in Angular Pivotview component	399
Tooltip Template.....	400
Css customization in Angular Pivotview component.....	402
Hiding Axis.....	402
Text Alignment.....	403
Customize header, value and summary cell style.....	404
Printing and Exporting	405
Print in Angular Pivotview component	405
Excel export in Angular Pivot Table component.....	407
Pdf export in Angular Pivotview component.....	426

Globalization and localization in Angular Pivotview component	457
Internationalization.....	457
Localization	462
Right-to-left (RTL).....	469
See Also	471
Accessibility in Angular Pivotview component	471
WAI-ARIA attributes.....	472
Keyboard interaction	473
Ensuring accessibility	479
See also	483
Ej1 api migration in Angular Pivotview component	484
Data Binding.....	484
Aggregation.....	485
Number Format.....	485
Summary Customization	486
Drill operation	487
Field List	487
Grouping Bar	488
Filtering	488
Maximum node limit in member editor	488
No Data Items	489
Advanced filtering.....	489
Drill Through	489
Cell Editing	490
Hyperlink.....	490
Defer Layout Update.....	492
Sorting.....	492
Value Sorting.....	493
Calculated Field.....	493
Paging.....	494
Conditional Formatting	494
Exporting.....	494
Grid Customization	495
Accessibility.....	496
Common.....	496

How To	498
Switching older themes style in Angular Pivotview component	498
Customize number date and time values in Angular Pivotview component.....	500
Customize the icons for pivot grid in Angular Pivotview component	502
PivotView_PivotFieldList .e-icons.e-toggle-field-list::before {.....	503
Configure data grid options on editing mode in Angular Pivotview component	504
Refresh the field list in Angular Pivotview component.....	505
Hide empty headers in Angular Pivotview component	507
Customizing loading indicator in Angular Pivotview component.....	508
Changing the pivotview component minimum height in Angular Pivotview component.....	509
Chart based on pivot table selection in Angular Pivotview component.....	510
Drill through grid cell edit type in Angular Pivotview component	513
Show field list when pivot table empty in Angular Pivotview component.....	515
Apply custom style to pivot cells in Angular Pivotview component.....	516
Show tooltip for row and column headers in Angular Pivotview component.....	518
Hide specific columns in Angular Pivotview component.....	520
Add custom aggregation type to the menu in Angular Pivotview component	523
Convert complex JSON to flat JSON and assign it to the pivot table in Angular Pivotview component	525
Load desired report from the report list as default in Angular Pivotview component	529
Display string value to pivot table values	532
Predefined Dialogs	534
Getting started with Angular Predefined dialogs component.....	534
Dependencies.....	534
Setup an Angular environment.....	534
Create an Angular application	534
Installing Syncfusion Popups package	534
Adding Dialog module.....	535
Adding CSS reference.....	536
Render a dialog using utility functions.....	536
Alert dialog.....	537
Confirm dialog.....	538
Prompt dialog.....	539
Draggable in Angular Predefined dialogs component	540
Alert dragging.....	540

Confirm drag	541
Prompt drag	542
Animation in Angular Predefined dialogs component.....	543
Alert animation	543
Confirm animation	543
Prompt animation	544
Position in Angular Predefined dialogs component	545
Alert position.....	545
Confirm position	546
Prompt position	547
Dimension in Angular Predefined dialogs component	547
Alert dimension.....	547
Confirm dimension.....	548
Prompt dimension	549
Max-width and max-height.....	550
Min-width and min-height	551
Customization in Angular Predefined dialogs component	552
Alert action button.....	552
Confirm action buttons	553
Prompt action buttons.....	553
Show or hide dialog close button	554
Alert dialog close button.....	554
Confirm dialog close button.....	555
Prompt dialog close button.....	556
Customize dialog content	557
ProgressBar	558
Getting started with Angular Progress bar component.....	558
Dependencies.....	558
Installation and Configuration	558
Create an Angular Application	558
Installing Syncfusion progressbar package	558
Registering progressbar Module.....	559
Types in Angular Progress bar component.....	560
Linear.....	560
Circular	561

Tooltip in Angular Progress bar component	562
Tooltip	562
Format.....	563
Customization	564
States in Angular Progress bar component	565
Determinate	565
Indeterminate	565
Buffer	566
Customization in Angular Progress bar component	567
Segments.....	567
Thickness	567
Min, Max, and Value	568
Radius.....	569
InnerRadius	570
Progress color and track color	570
Annotation in Angular Progress bar component	571
Annotation	571
Label.....	572
Animation in Angular Progress bar component.....	573
Range in Angular Progress bar component	574
Events in Angular Progress bar component.....	575
Value Change	575
ProgressCompleted.....	576
Accessibility in Angular Progress bar component.....	576
WAI-ARIA attributes.....	577
Keyboard interaction	578
Ensuring accessibility	578
See also	578
ProgresButton	578
Getting started with Angular Progress button component	578
Dependencies.....	578
Setup Angular environment.....	578
Create an Angular application	579
Installing Syncfusion ProgresButton package	579
Adding ProgresButton module	580

Adding Syncfusion ProgressButton component	580
Adding CSS reference.....	580
Running the application	581
Enable progress in button.....	581
Spinner and progress in Angular Progress Button component	582
Change spinner position	582
Progress.....	583
See Also	586
Accessibility in Angular Progress button component	586
WAI-ARIA attributes.....	587
Keyboard interaction	588
Ensuring accessibility	588
See also	588
How To	588
Change the text content and styles of the progressbutton during progress in Angular Progress button component.....	588
Customize progress using cssclass in Angular Progress button component	589
Hide spinner in Angular Progress button component	590
Trace events of progress button in Angular Progress button component	590
QueryBuilder	592
Getting started with Angular Query builder component	592
Dependencies.....	592
Setup Angular environment.....	592
Create an Angular application	592
Installing Syncfusion Query Builder package	592
Adding Query Builder module.....	593
Adding Syncfusion Query Builder component.....	593
Adding CSS reference.....	594
Running the application	594
Rendering with rule.....	595
Columns in Angular Query builder component	596
Auto generation	597
Labels	597
Operators	597
Step	598

Format.....	598
Validations	600
Data binding in Angular Query builder component.....	601
Local data	601
Remote data.....	602
Data Manager	608
Complex Data Binding.....	610
Filtering in Angular Query builder component	612
Template in Angular Query builder component.....	614
Header Template	614
Column Template.....	616
Rule Template	620
Model binding in Angular Query builder component.....	623
Importing and Exporting in Angular Query builder component	624
Importing	624
Exporting.....	633
Lock Group/Rule in Angular Query builder component	640
Clone Group/Rule in Angular Query builder component	642
Global local in Angular Query builder component	644
Style and appearance in Angular Query builder component	647
Accessibility in Angular Query Builder component	647
WAI-ARIA attributes.....	648
Keyboard interaction	649
Ensuring accessibility	649
See also	649
How To	649
Display mode in Angular Query builder component	649
Sort columns in Angular Query builder component	651
Restrict groups in Angular Query builder component.....	652
State persistence in Angular Query builder component	653
Rtl in Angular Query builder component.....	655
Summary view in Angular Query builder component	656
RadioButton	658
Getting started with Angular Radio button component.....	658
Dependencies.....	658

Setup Angular environment.....	658
Create an Angular application	659
Installing Syncfusion RadioButton package	659
Adding RadioButton module.....	660
Adding Syncfusion RadioButton component	660
Adding CSS reference.....	660
Running the application	661
Change the RadioButton state	661
Label and size in Angular Radio button component.....	662
Label.....	662
Size	663
See Also	664
Accessibility in Angular Radio button component.....	664
WAI-ARIA attributes.....	665
Keyboard interaction	665
Ensuring accessibility	665
See also	665
How To	665
Customize radiobutton appearance in Angular Radio button component	665
Name and value in form submit in Angular Radio button component	666
Right to left in Angular Radio button component	667
Set the disabled state in Angular Radio button component.....	668
Two way binding using radiobutton in Angular Radio button component	669
Ej1 api migration in Angular Radio button component	671
Properties.....	671
Methods.....	672
Events.....	672
Range Navigator.....	673
Getting started with Angular Range navigator component.....	673
Setup Angular Environment.....	673
Create an Angular Application	673
Installing Syncfusion Chart package.....	673
Registering RangeNavigator Module	674
Module Injection.....	675
Populate Range Navigator with Data.....	676

Enable Tooltip	677
Selecting range in Angular Range navigator component.....	678
Lightweight in Angular Range navigator component	679
See Also	680
Series types in Angular Range navigator component	680
Line	680
Area	681
StepLine.....	682
Axis in Angular Range navigator component.....	683
Numeric.....	683
Logarithmic Axis	687
Date-time	690
Period selector in Angular Range navigator component	693
Periods	693
Positioning period selector	695
Height.....	696
Visibility of range navigator	697
See Also	698
Labels in Angular Range navigator component	698
Multilevel labels	698
Grouping	699
Smart labels.....	700
Label positioning.....	701
Labels customization.....	702
Grid tick in Angular Range navigator component.....	703
Grid line customization	703
Tick line customization.....	704
Customization in Angular Range navigator component	705
Navigator appearance.....	705
Thumb	706
Border customization.....	707
Deferred update.....	708
Allow snapping.....	709
Animation.....	710
See Also	711

Tool tip in Angular Range navigator component	711
Customization	711
Label Format	712
R t l in Angular Range navigator component	713
Export print in Angular Range navigator component	714
Export.....	714
Print.....	715
Accessibility in Angular Range navigator component.....	716
WAI-ARIA attributes.....	717
Keyboard interaction	717
Ensuring accessibility	718
See also	718
Ej1 api migration in Angular Range navigator component	718
RangeNavigator.....	718
Series.....	721
StyleSettings.....	722
Tooltip	723
Period Selector.....	724
Methods	724
Events.....	724
Range Slider	726
Getting started with Angular Range Slider component.....	726
Setting up angular project	726
Create a new application	726
Installing Syncfusion RangeSlider Package	726
Adding Slider Module.....	727
Adding Syncfusion Slider Component.....	728
Adding Slider CSS reference.....	729
Types	730
Customization	731
See Also	733
Schematics in Angular Range slider component.....	733
Getting started.....	733
Dependency and Module injection using Schematics	734
Component generation using Schematics	734

Ticks in Angular Range slider component	735
Step	736
Min and Max	736
Format in Angular Range slider component	737
Using format API	738
Using Events	739
Limits in Angular Range slider component	740
Default and MinRange Slider limits	740
Range Slider limits	741
Handle lock	742
Style in Angular Range slider component	743
Customizing the slider track	743
Customizing the slider handle	743
Customizing the slider limits	744
Customizing the slider ticks	744
Customizing the slider buttons	744
Accessibility in Angular Range Slider component	744
WAI-ARIA attributes	745
Keyboard interaction	746
Ensuring accessibility	746
See also	746
Ej1 api migration in Angular Range slider component	746
How To	748
Customize slider bar in Angular Range slider component	748
Customize slider limits in Angular Range slider component	751
Customize slider thumb in Angular Range slider component	753
Customize slider ticks label in Angular Range slider component	755
Date range slider in Angular Range slider component	756
Form slider with formvalidator in Angular Range slider component	757
Numeric range slider in Angular Range slider component	761
Slider in angular reactive form in Angular Range slider component	762
Slider validation using template driven forms in Angular Range slider component	765
Slider with ngmodel in Angular Range slider component	767
Time range slider in Angular Range slider component	768
Customize slider msword in Angular Range slider component	769

slider.e-control.e-slider .e-handle {	770
Show slider from hidden state in Angular Range slider component	772
Reversible Range Slider in Angular	773
Rating	775
Getting started with Angular Rating component.....	775
Dependencies.....	775
Setup Angular environment.....	775
Create an Angular application	775
Installing Syncfusion Rating package	775
Adding Rating module.....	776
Adding Syncfusion Rating component.....	777
Adding CSS reference.....	777
Running the application.....	777
Value	778
Precision Modes in Angular Rating Component	778
Labels in Angular Rating Component.....	780
Label position	780
Label template	781
Tooltip in Angular Rating Component	782
Tooltip template	782
Tooltip customization	783
Selection in Angular Rating Component.....	785
Min value	785
Single selection	786
Show or hide reset button	787
Empty (unrated) symbol template.....	787
Full (rated) symbol template	790
Using Emoji icon as rating symbol	792
Using SVG icon as rating symbol.....	793
Using PNG image as rating symbol	795
Events in Angular Rating Component	796
beforeItemRender	796
created	796
onItemHover	797
valueChanged.....	797

Appearance in Angular Rating Component	799
Items count	799
Disabled.....	799
Visible.....	800
Read only.....	801
CssClass	801
Changing icon using CssClass	804
Accessibility in Angular Rating component.....	806
WAI-ARIA attributes.....	807
Keyboard interaction	807
Ensuring accessibility	808
See also	808
Ribbon	808
Getting started with Angular Ribbon component	808
Dependencies.....	808
Setup Angular environment.....	809
Create an Angular application	809
Installing Syncfusion Ribbon Package	809
Adding Ribbon module	810
Adding CSS reference.....	810
Adding Syncfusion Ribbon component.....	811
Adding Ribbon Tab	811
Adding Ribbon Group.....	811
Adding Ribbon Item	812
Running the application.....	813
Modules in Ribbon component	818
Tabs and Groups	819
Adding Tabs.....	819
Adding Groups	820
Adding Items	821
Items in Angular Ribbon component	822
Built-in items.....	822
Custom items	841
Items display Mode.....	843
Enable or disable items	847

Layouts in Angular Ribbon component.....	848
Classic layout.....	848
Simplified layout	864
Minimized state	868
Show or hide the layout switcher	869
File Menu	871
Visibility	871
Adding menu items	872
Open submenu on click.....	874
Custom header text	875
Ribbon Backstage.....	877
Adding backstage items	877
Adding footer items	879
Adding separator.....	882
Back button	884
Backstage target.....	886
Template	888
Setting width and height.....	891
Ribbon Contextual Tabs	893
Visible tabs	893
Adding contextual tabs	893
Selected tabs.....	896
Methods	897
Ribbon Keytip	899
Ribbon items keytip	899
File menu keytip.....	903
Backstage menu keytip	905
Ribbon layout switcher keytip	909
Ribbon launcher icon keytip	910
Methods	912
Guidelines for adding keytips.....	912
Gallery Items in Angular Ribbon component.....	913
Groups.....	913
Setting item count.....	931
Setting selected item	933

Setting popup height.....	935
Setting popup width.....	935
Help Pane	937
Tooltip	938
Adding title.....	938
Adding content.....	940
Adding icon	941
Customization	943
Ribbon Resizing	945
Defining items allowed size	945
Defining items active size.....	947
Events.....	947
tabSelected	947
tabSelecting.....	948
ribbonCollapsing	949
ribbonExpanding	950
launcherIconClick	951
Button item events	952
CheckBox item events.....	954
ComboBox item events	963
DropDown item events	971
SplitButton item events	979
GroupButton item events	987
FileMenu events.....	990
Backstage Menu events	998
Gallery events	1000
Accessibility in Angular Ribbon component	1009
WAI-ARIA attributes.....	1010
Keyboard interaction	1011
Ensuring accessibility	1012
See also	1012
RichTextEditor.....	1012
Getting started with Angular Rich text editor component.....	1012
Setup Angular Environment.....	1012
Create an Angular Application	1013

Installing Syncfusion Rich Text Editor package	1013
Adding Rich Text Editor module	1014
Adding CSS reference.....	1014
Adding Rich Text Editor component	1015
Initialize Rich Text Editor from ` <code><iframe></code> ` element.....	1016
Module Injection.....	1017
Run the application	1018
Configure the Toolbar	1019
Insert images and links.....	1020
Retrieve the formatted content.....	1021
Retrieve the number of characters in the Rich Text Editor	1022
See Also	1022
Editor mode in Angular Rich text editor component.....	1022
HTML Editor	1022
Markdown Editor	1023
Toolbar in Angular Rich text editor component	1025
Expand Toolbar	1025
Multi-row Toolbar	1026
Floating Toolbar	1027
Toolbar Items	1028
Custom tool.....	1033
Quick inline toolbar	1036
Styling in Angular Rich text editor component	1038
Font name and Font size	1038
Custom fonts and size	1039
Font and Background color	1040
Editor content styles	1041
Image in Angular Rich text editor component.....	1047
Upload options.....	1048
Delete Image	1049
Insert from web	1051
Dimension	1052
Caption and Alt Text.....	1052
Display position.....	1052
Image with link.....	1053

Resize	1054
Drag and Drop	1054
Audio in Angular Rich text editor component	1057
Configure audio tool in the toolbar	1057
Insert audio from the web	1059
Insert audio from local machine	1059
Replacing audio	1064
Delete audio	1064
Display position	1065
Rename audio before inserting	1066
Upload audio with authentication	1069
See Also	1070
Video in Angular Rich text editor component	1070
Configure the video tool in the toolbar	1070
Insert a video from the web	1072
Insert video from local machine	1073
Replacing video	1078
Delete video	1079
Dimension	1080
Display position	1081
Resize video	1081
Rename video before inserting	1082
Upload video with authentication	1085
See Also	1086
Link in Angular Rich text editor component	1086
Insert link	1086
Remove Link	1087
Auto-link	1087
Manipulation	1087
Table in Angular Rich text editor component	1088
Insert table	1089
Quick Toolbar	1090
Table Header	1090
Insert Rows	1090
Insert Columns	1091

Set Color	1091
Delete Table	1091
Vertical Align	1091
Horizontal Align.....	1091
Table Styles	1092
Table Properties	1092
Table cell merge and split	1093
Emoji Picker in Angular RichTextEditor component	1095
Enabling the toolbar option and custom emojis.....	1095
Using the shortcut key to open the emoji picker.....	1099
Navigating and selecting emojis using the keyboard.....	1099
Markdown in Angular Rich text editor component	1099
Supported Commands	1100
Markdown to HTML	1101
Table.....	1104
Custom formation	1111
File browser in Angular Rich text editor component.....	1113
Required additional dependency.....	1114
Additional CSS Reference.....	1114
Format Painter in Angular Rich Text Editor Component Syncfusion	1116
Enabling the toolbar option for Format Painter	1116
Customization of copy and paste format.....	1117
Using the shortcut key to copy and paste the format	1120
Validation in Angular Rich text editor component	1120
Template driven forms.....	1120
Reactive forms	1122
Globalization in Angular Rich text editor component	1123
Localization	1123
RTL.....	1136
Miscellaneous in Angular Rich text editor component.....	1137
Placeholder	1137
Character count	1137
Code view.....	1138
Undo/redo manager	1141
Prevention of cross-site scripting (XSS)	1142

Resizable support	1144
Number and Bullet Format Lists	1146
Xhtml validation in Angular Rich text editor component	1147
Attributes	1147
HTML Elements	1147
Inline mode in Angular Rich text editor component	1148
Show on select/click.....	1148
Paste cleanup in Angular Rich text editor component	1149
MS Word to HTML	1149
Paste cleanup	1149
Prompt dialog.....	1150
Paste as plain text	1150
Keep format	1150
Denied tags	1150
Denied attributes	1151
Allowed style properties	1151
Mention integration in Angular Rich text editor component	1153
See Also	1155
Enter key in Angular Rich text editor component	1155
Enter key customization.....	1155
Shift-Enter key customization	1157
Iframe in Angular Rich text editor component	1159
IFrame attributes	1160
Adding external CSS/Script file	1161
Third party integration in Angular Rich text editor component	1162
CodeMirror Integration.....	1162
Integrate `embedly`	1165
Exec command in Angular Rich text editor component	1167
Style in Angular Rich text editor component.....	1168
Customizing the Rich Text Editor's content	1168
Customizing the Rich Text Editor's toolbar	1169
Customizing the Rich Text Editor's character count	1169
Keyboard support in Angular Rich text editor component.....	1170
HTML formation shortcut key.....	1170
Markdown formation shortcut key.....	1172

Custom key config.....	1174
Accessibility in Angular Rich text editor component	1175
ARIA attributes.....	1176
Keyboard interaction	1177
Ensuring accessibility	1179
See also	1179
How To	1179
Add google fonts in Angular Rich text editor component	1179
Change default font family in Angular Rich text editor component.....	1181
Check image size in Angular Rich text editor component	1183
Customize placeholder style in Angular Rich text editor component	1184
Customize shortcut keys in Angular Rich text editor component	1184
Set the cursor at the specific range in Angular Rich text editor component.....	1186
Update value in Angular Rich text editor component	1187
Rename images in server in Angular Rich text editor component	1188
File attachment in Angular Rich text editor component	1191
Format code block in Angular Rich text editor component.....	1195
Schedule.....	1196
Getting started with Angular Schedule component	1196
Setup Angular Environment.....	1196
Create an Angular Application	1196
Installing Syncfusion Schedule package.....	1196
Registering Schedule Module	1197
Adding CSS reference.....	1198
Module injection.....	1198
Initialize the Schedule	1199
Populating appointments	1199
Setting date.....	1201
Setting view.....	1202
Individual view customization	1203
Module injection in Angular Schedule component	1203
Module injection.....	1204
Scheduler interactions in Angular Schedule component.....	1204
Appointments in Angular Schedule component.....	1206
Normal events.....	1206

Spanned events.....	1207
All-day events.....	1207
Customize the rendering of the spanned events.....	1210
Recurring events	1212
Event fields.....	1220
Adding Custom fields	1224
Customize the order of the overlapping events	1225
Drag and drop appointments.....	1227
Inline Appointment	1240
Appointment Resizing	1241
Appointment customization	1246
Setting minimum height	1250
Block Dates and Times	1251
Readonly	1254
Make specific events readonly.....	1255
Restricting event creation on specific time slots	1256
Differentiate the past time events.....	1258
Appointments occupying entire cell	1259
How to limit maximum number of events to display	1260
Display tooltip for appointments.....	1262
Appointment filtering	1265
Appointment selection	1267
Deleting multiple appointments.....	1267
Retrieve event details from the UI of an event	1267
Get the current view appointments	1269
Get the entire appointment collections.....	1271
Refresh appointments	1273
Data binding in Angular Schedule component	1273
Binding local data.....	1274
Binding remote data	1275
Loading data via AJAX post	1279
Passing additional parameters to the server	1280
Handling failure actions	1281
Scheduler CRUD actions.....	1282
Configuring Scheduler with Google API service	1286

Crud actions in Angular Schedule component.....	1288
Add	1288
Edit	1293
Delete.....	1305
Drag and drop	1313
Resize	1314
Virtual scrolling in Angular Schedule component.....	1315
Enabling lazy loading for appointments.....	1318
See Also	1321
Editor template in Angular Schedule component	1321
Event editor.....	1321
Customizing event editor using template	1334
Quick popups	1356
More events indicator and popup	1362
Timezone in Angular Schedule component	1372
Understanding date manipulation in JavaScript.....	1372
Scheduler with no timezone	1372
Scheduler set to specific timezone	1373
Display events on same time everywhere with no time difference	1374
Set specific timezone for events	1375
Add or remove timezone names to/from the timezone collection.....	1376
Timezone methods	1377
Views in Angular Schedule component	1380
Setting specific view on scheduler	1380
View specific configuration	1382
Extending view intervals	1402
See Also	1403
Calendar mode in Angular Schedule component	1403
Gregorian Calendar	1403
Islamic Calendar	1404
Resources in Angular Schedule component	1405
Resource fields.....	1406
Resource data binding	1406
Scheduler with multiple resources	1409
Resource grouping	1411

Customizing parent resource cells	1422
Working with shared events	1424
Simple resource header customization	1426
Customizing resource header with multiple columns	1429
Collapse/Expand child resources in timeline views	1431
Displaying tooltip for resource headers.....	1433
Choosing among resource colors for appointments.....	1434
Dynamically add and remove resources	1437
Setting different working days and hours for resources	1439
Hide non-working days when grouped by date	1443
Compact view in mobile.....	1445
Adaptive UI in desktop.....	1446
Header rows in Angular Schedule component	1449
Display year and month rows in timeline views	1451
Display week numbers in timeline views.....	1452
Timeline view displaying dates of a complete year	1454
Customizing the header rows using template	1455
Row auto height in Angular Schedule component	1457
Calendar month view	1458
Timeline views.....	1459
Timeline views with multiple resources	1461
Appointments occupying entire cell	1462
Header bar in Angular Schedule component.....	1464
Show or Hide header bar	1464
Customizing header bar	1466
How to display the view options within the header bar popup	1468
Date header customization.....	1469
Customizing the date range text.....	1473
Customizing header indent cells	1474
Timescale in Angular Schedule component.....	1476
Setting different time slot duration	1476
Customizing time cells using template	1477
Hide the timescale	1479
Highlighting current date and time.....	1480
Working days in Angular Schedule component.....	1482

Set working days	1482
Hiding weekend days	1484
Show week numbers.....	1485
Set working hours	1489
Scheduler displaying custom hours	1490
Setting start day of the week	1491
Scroll to specific time and date.....	1493
See Also	1495
Cell customization in Angular Schedule component	1496
Setting cell dimensions in all views.....	1496
Check for cell availability.....	1497
Customizing cells in all the views	1499
Customizing cell header in month view	1503
Customizing the minimum and maximum date values	1504
Customizing the weekend cells background color.....	1505
How to disable multiple cell and row selection in Schedule	1507
State persistence in Angular Schedule component	1507
Exporting in Angular Schedule component	1508
Excel Exporting.....	1508
Exporting calendar events as ICS file	1520
Import events from other calendars.....	1523
How to print the Scheduler element	1524
Context menu in Angular Schedule component.....	1527
Dimensions in Angular Schedule component	1531
Auto Height and Width	1531
Height and Width in pixel	1532
Height and Width in percentage.....	1533
See Also	1534
Recurrence editor in Angular Schedule component.....	1534
Customizing the repeat type option in editor	1534
Customizing the End Type Option in Editor	1536
Accessing the recurrence rule string.....	1537
Set specific value on recurrence editor	1538
Recurrence date generation	1539
Recurrence date generation in server-side.....	1540

Restrict date generation with specific count	1540
Localization in Angular Schedule component	1542
Globalization	1542
Localizing the static Scheduler text.....	1546
Setting date format.....	1551
Setting the time format	1553
Displaying Scheduler in RTL mode	1554
See Also	1556
Accessibility in Angular Schedule component	1556
ARIA attributes.....	1557
Keyboard interaction	1558
Ensuring accessibility	1559
See also	1559
Scheduler styling in Angular Schedule component	1559
Frequently asked questions in Angular Schedule component	1561
Maximum call stack size exceeded	1561
Grouping with empty resources	1562
Not providing e-field in editor template.....	1562
Missing CSS reference	1562
QuickInfoTemplate at bottom	1563
Not importing culture files while using localization	1564
Ej1 api migration in Angular Schedule component.....	1565
Scheduler	1565
Properties.....	1565
Methods.....	1575
Events.....	1578
How To	1582
Add edit and remove events in Angular Schedule component	1582
Set default value for event fields in Angular Schedule component	1585
Open event editor manually in Angular Schedule component.....	1587
Prevent date navigation in Angular Schedule component	1589
Half yearly view in Angular Schedule component	1590
Set different work hours in Angular Schedule component	1592
Show tool tip with delay in Angular Schedule component.....	1593
Render schedule inside dialog in Angular Schedule component.....	1594

Quick info template in Angular Schedule component	1596
Enable scroll option on all day section in Angular Schedule component	1600
Manual refresh in Angular Schedule component	1602
Sidebar	1608
Getting started with Angular Sidebar component.....	1608
Prerequisites	1608
Setting up angular project	1609
Adding Dependencies	1609
Installing Syncfusion Sidebar Package	1609
Adding Styles.....	1610
Adding Sidebar module.....	1610
Adding Syncfusion component	1611
Run the application	1612
Enable backdrop	1613
Position	1614
Animate.....	1616
Close on document click	1617
Enable gestures.....	1618
See Also	1619
Custom context in Angular Sidebar component.....	1619
See Also	1621
Types in Angular Sidebar component	1621
Expanding types of Sidebar	1621
See Also	1623
Docking sidebar in Angular Sidebar component.....	1623
See Also	1625
Auto close in Angular Sidebar component	1625
Style in Angular Sidebar component	1627
Customizing the sidebar.....	1627
Customizing the sidebar based on the positions	1627
Customizing the sidebar based on the active state	1627
Customizing the sidebar with dock state.....	1628
Customizing the different types of sidebar.....	1628
Customizing the backdrop of the sidebar	1629
Customizing the sidebar in the RTL direction	1629

Prevent the animation transition for the Sidebar component	1629
Accessibility in Angular Sidebar component.....	1630
WAI-ARIA attributes.....	1631
Keyboard interaction	1631
Ensuring accessibility	1631
See also	1631
How To	1631
Initialize sidebar using systemjs in Angular Sidebar component.....	1631
Initialize the sidebar listview in Angular Sidebar component	1636
Open and close the sidebar in Angular Sidebar component	1637
Top and bottom sidebar in Angular Sidebar component	1639
Multiple sidebar in Angular Sidebar component.....	1642
Sidebar with treeview in Angular Sidebar component.....	1643
Fixed sidebar in Angular Sidebar component.....	1647
Sidebar with variation animation in Angular Sidebar component	1653
Hide sidebar in Angular Sidebar component.....	1655

Pivot Table

Getting started with Angular Pivotview component

This section explains you the steps required to create a simple pivot table and demonstrate the basic usage of the [pivot table component](#)[Link to the Video](#) in an Angular environment.

To get start quickly with Angular Pivot Table, you can check on this video:

Setup Angular Environment

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

To install Angular CLI use the following command.

```
`bash
npm install -g @angular/cli
`
```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
ng new my-app
cd my-app
`
```

Dependencies

The following list of dependencies are required to use the pivot table component in your application.

```
`javascript
|-- @syncfusion/ej2-angular-pivotview
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-pivotview
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-excel-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-inputs
```

```
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-angular-base
\
```

Installing Syncfusion PivotView package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(`>=20.2.36`) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-pivotview](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-pivotview --save
\
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-pivotview@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-pivotview@ngcc --save
\
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-pivotview:"20.2.38-ngcc"
\
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Registering PivotView Module

Import PivotView module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-pivotview` [src/app/app.module.ts].

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the PivotViewModule for the pivot table component
import { PivotViewModule } from '@syncfusion/ej2-angular-pivotview';
import { AppComponent } from './app.component';
@NgModule({
//declaration of ej2-angular-pivotview module into NgModule
imports: [ BrowserModule, PivotViewModule ],
declarations: [ AppComponent ],
bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding CSS reference

The following CSS files are available in `../node_modules/@syncfusion` package folder. This can be referenced in [src/styles.css] using following code.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/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-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-pivotview/styles/material.css';
```

Browser compatibility

Polyfills are required to use the Pivot Table in Internet Explorer 11 browser. Refer the [documentation](#) for more details.

Initializing pivot table component in an application

Add the template in [src/app/app.component.ts] file to render the pivot table component.

Add the Angular pivot table by using `<ejs-pivotview>` selector in `template` section of the `app.component.ts` file.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
  }
}
`
```

Assigning sample data to the pivot table

The Pivot Table component further needs to be populated with an appropriate data source. For illustration purpose, a collection of objects mentioning the sales details of certain products over a period and region has been prepared. This sample data is assigned to the pivot table component through `dataSource` property under [dataSourceSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
```

```

public dataSourceSettings: IDataOptions;

ngOnInit(): void {
  this.pivotData = [
    { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
    { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
    { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
    { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
    { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }
  ];
  this.dataSourceSettings = {
    dataSource: this.pivotData,
  };
}

```

Adding fields to row, column, value and filter axes

Now that pivot table is initialized and assigned with sample data, will further move to showcase the component by organizing appropriate fields in row, column, value and filter axes.

In [dataSourceSettings](#), four major axes - [rows](#), [columns](#), [values](#) and [filters](#) plays a vital role in defining and organizing fields from the bound data source, to render the entire pivot table component in a desired format.

[rows](#) – Collection of fields that needs to be displayed in row axis of the pivot table.

[columns](#) – Collection of fields that needs to be displayed in column axis of the pivot table.

[values](#) – Collection of fields that needs to be displayed as aggregated numeric values in the pivot table.

[filters](#) - Collection of fields that would act as master filter over the data bound in row, column and value axes of the pivot table.

In-order to define each field in the respective axis, the following basic properties should be set.

- [name](#): It allows to set the field name from the bound data source. It's casing should match exactly like in the data source and if not set properly, the pivot table will not be rendered.
- [caption](#): It allows to set the field caption, which is the alias name of the field that needs to be displayed in the pivot table.
- [type](#): It allows to set the summary type of the field. By default, **Sum** is applied.

In this illustration, "Year" and "Quarter" are added in column, "Country" and "Products" in row, and "Sold" and "Amount" in value section respectively.

`typescript

```
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.pivotData = [
      { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
      { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
      { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
      { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
      { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }];
    this.dataSourceSettings = {
      dataSource: this.pivotData,
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }]
    };
  }
}
```

Applying formatting to a value field

Formatting defines a way in which values should be displayed. For example, format “C” denotes the values should be displayed in currency pattern. To do so, define the [formatSettings](#) with its [name](#) and [format](#) properties and add it to [formatSettings](#). In this illustration, the [name](#) property is set as **Amount**, a field from value section and its format is set as currency. Likewise, we can set format for other value fields as well and add it to [formatSettings](#).

Only fields from value section, which is in the form of numeric data values are applicable for formatting.

`typescript

```
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.pivotData = [
      { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
      { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
      { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
      { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
      { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products': 'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }
    ];
    this.dataSourceSettings = {
      dataSource: this.pivotData,
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    };
  }
}
```

```

values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
rows: [{ name: 'Country' }, { name: 'Products' }],
formatSettings: [{ name: 'Amount', format: 'C0' }],
filters: []
};
}
}
`

```

Module Injection

To create pivot table with additional features, inject the required modules. The modules that are available with basic functionality are as follows.

- **GroupingBarService** - Inject this module to access grouping bar feature.
- **FieldListService** - Inject this module to access pivot field list feature.
- **CalculatedFieldService** - Inject this module to access calculated field feature.

These modules should be injected into the **providers** section of root **NgModule** or component class. On doing so, only the injected views will be loaded and displayed along with pivot table.

```

`typescript
providers: [GroupingBarService]
`

```

Enable Field List

The field list allows to add or remove fields and also rearrange the fields between different axes, including column, row, value, and filter along with filter and sort options dynamically at runtime. It can be enabled by setting the [showFieldList](#) property to **true** and by injecting the **FieldListService** module as follows. To know more about field list, [refer](#) here.

If the **FieldListService** module is not injected, the Field List will not be rendered with the pivot table component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width showFieldList='true'></ejs-pivotview></div>`
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.width = "100%";
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Enable Grouping Bar

The grouping bar feature automatically populates fields from the bound data source and allows end users to drag fields between different axes such as columns, rows, values, and filters, and alter pivot table at runtime. It also provides option to sort, filter and remove fields. It can be enabled by setting the [showGroupingBar](#) property to **true** and by injecting the **GroupingBarService** module as follows. To know more about grouping bar, [refer](#) here.

If the **GroupingBarService** module is not injected, the grouping bar will not be rendered with the pivot table component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';

```

```

import { IDataOptions, IDataset, PivotView, GroupingBarService } from
 '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showGroupingBar='true'></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = "100%";
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Exploring Filter Axis

The filter axis contains collection of fields that would act as master filter over the data bound in row, column and value axes of the pivot table. The fields along with filter members could be set to filter axis either through report via code behind or by dragging and dropping fields from other axes to filter axis via grouping bar or field list at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

],
standalone: true,
selector: 'app-container',
providers: [GroupingBarService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showGroupingBar='true'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [{ name: 'Country' }]
        };
        this.width = "100%";
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Calculated field

The calculated field feature allows user to insert or add a new calculated field based on the available fields from the bound data source using basic arithmetic operators. The calculated field can be included in pivot table using the [calculatedFieldSettings](#) from code behind. Or else, calculated fields can be added at run time through the built-in dialog by just setting the [allowCalculatedField](#) property to **true** in pivot table. You will see a button enabled in the Field List UI automatically to invoke the calculated field dialog and perform necessary operation. To know more about calculated field, [refer](#) here.

Also calculated fields can be added to the bound datasource at run time through the built-in popup. The popup can be enabled by setting the `allowCalculatedField` property to true and by injecting the `CalculatedFieldService` module as follows.

If the `CalculatedFieldService` module is not injected, the calculated field popup will not be rendered with the pivot table component. Moreover calculated field is applicable only for value fields.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
CalculatedFieldService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService, CalculatedFieldService],
// specifies the template string for the pivot table component
template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width allowCalculatedField='true' showFieldList='true'></ejs-
pivotview></div>`
})
export class AppComponent {
public dataSourceSettings?: IDataOptions;
public width?: string;
ngOnInit(): void {
this.dataSourceSettings = {
dataSource: Pivot_Data as IDataset[],
expandAll: false,
enableSorting: true,
drilledMembers: [{ name: 'Country', items: ['France'] }],
columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }, { name: 'Total', caption: 'Total
Amount', type: 'CalculatedField' }],
rows: [{ name: 'Country' }, { name: 'Products' }],
formatSettings: [{ name: 'Amount', format: 'C0' }],
filters: [],
calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
};
this.width = "100%";
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Run the application

The quickstart project is configured to compile and run the application in the browser. Use the following command to run the application.

```
`sh
ng serve --open
`
```

To run the application in production mode, set the **buildOptimizer** to **false** in **angular.json** to resolve the angular-cli problem.

```
`sh
ng serve --prod
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
CalculatedFieldService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, CalculatedFieldService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width allowCalculatedField='true' showFieldList='true'></ejs-
pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
```



```

        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount) "+"Sum(Sold) "' }]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD028 -->

In Angular, the `ViewChild` method is used to access the instance of the component. It has following parameters.

- * `selector` - The directive type or the name used for querying.
- * `read` - Read a different token from the queried elements.
- * `static` - `true` to resolve query results before change detection runs, `false` to resolve after change detection. Default is `false`.

Under Angular 8 version, the `static` parameter is optional. So, follow the below code.

```
@ViewChild('pivotview')
```

But the parameter is mandatory in Angular 8 and above versions. So, follow the below code.

```
@ViewChild('pivotview', { static: false })
```

You can also explore our [Angular Pivot Table example](#) [Link to the Video](#) that shows how to rendering of the pivot table with drill-up and drill-down functionality bound to a relational report.

Data binding in Angular Pivotview component

To get start quickly with Data Binding, you can check on this video:

JSON

For JSON data binding, the `type` property under `dataSourceSettings` needs to be set as `JSON`. By default, the default value is assumed as `JSON`.

Binding JSON data via local

In-order to bind local JSON data to the pivot table user can assign the local variable holding the JSON data to the `dataSource` property under `dataSourceSettings`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Using local variable, the JSON data can also be bound to the pivot table using [DataManager](#) option with the help of [JsonAdaptor](#). Here the instance of [DataManager](#) holding JSON data is assigned to [dataSource](#) property under [dataSourceSettings](#). The use of [DataManager](#) is optional here.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { DataManager, JsonAdaptor } from '@syncfusion/ej2-data';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
public dataSourceSettings?: IDataOptions;
public data?: DataManager;
public width?: string;
ngOnInit(): void {
    this.data = new DataManager({
        json: Pivot_Data,
        // tslint:disable-next-line:new-parens
        adaptor: new JsonAdaptor
    });
    this.dataSourceSettings = {
        dataSource: this.data,
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

In the meantime, the JSON data from the local *.json file type can also be connected to the pivot table via the file uploader option. Here, the resulting string after uploading the file needs to be converted to JSON data that can be assigned to the [dataSource](#) property under [dataSourceSettings](#). The following code example illustrates the same.

```
`javascript
import { Component, OnInit } from '@angular/core';
import { IDataOptions } from '@syncfusion/ej2-angular-pivotview';
import { Uploader } from '@syncfusion/ej2-inputs';
@Component({
  selector: 'app-container',
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions;
  public width: string;
  public uploadObj: Uploader;
  ngOnInit(): void {
    // Step 1: Initiate the file uploader
    this.uploadObj: Uploader = new Uploader({
    });
    this.uploadObj.appendTo('#fileupload');
    let input = document.querySelector('input[type="file"]');
    // Step 2: Add the event listener which fires when the *.JSON file is uploaded.
    input.addEventListener('change', function (e: Event) {
    // Step 3: Initiate the file reader
    let reader: FileReader = new FileReader();
    reader.onload = function () {
    // Step 4: Getting the string output which is to be parsed as JSON.
    let result: any =JSON.parse(reader.result as string);
    this.dataSourceSettings = {
    // Step 5: The JSON result to be bound as data source.
    dataSource: result
    // Step 6: The appropriate report needs to be provided here.
    };
    };
    reader.readAsText((input as any).files[0]);
```

```

}
}
`

```

Binding JSON data via remote

In-order to bind remote JSON data, mention the endpoint [URL](#) under [dataSourceSettings](#) property. The [URL](#) property supports both direct downloadable file (*.json) and web service URL.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      url: 'https://cdn.syncfusion.com/data/sales-analysis.json',
      expandAll: false,
      rows: [
        { name: 'EnerType', caption: 'Energy Type' }
      ],
      columns: [
        { name: 'EneSource', caption: 'Energy Source' }
      ],
      values: [
        { name: 'PowUnits', caption: 'Units (GWh)' },
        { name: 'ProCost', caption: 'Cost (MM)' }
      ],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

CSV

For CSV data binding, the **type** property under [dataSourceSettings](#) needs to be set as **CSV** mandatorily.

The CSV format is considered to be the most compact format compared to JSON since it is half the size of JSON. This helps to reduce the bandwidth while transferring to the browser.

Binding CSV data via local

In-order to bind local CSV data to the pivot table, user needs to convert it as string array and then directly assign it to the [dataSource](#) property under [dataSourceSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { csvdata } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: this.getCSVData(),
      type: 'CSV',
      expandAll: false,
      enableSorting: true,
      // tslint:disable-next-line:max-line-length
      formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name: 'Total
Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
      drilledMembers: [{ name: 'Item Type', items: ['Baby Food' ]}],
      rows: [
        { name: 'Region' },
        { name: 'Country' }
      ],
      columns: [
        { name: 'Item Type' },
        { name: 'Sales Channel' }
      ],
      values: [
        { name: 'Total Cost' },
```

```

        { name: 'Total Revenue' },
        { name: 'Total Profit' }
    ],
    filters: []
};
this.width = '100%';
}
getCSVData(): string[][] {
    const dataSource: string[][] = [];
    const jsonObject: string[] = csvdata.split(/\r?\n|\r/);
    // tslint:disable-next-line:prefer-for-of
    for (let i = 0; i < jsonObject.length; i++) {
        if (!isNullOrUndefined(jsonObject[i]) && jsonObject[i] !== '') {
            dataSource.push(jsonObject[i].split(','));
        }
    }
    return dataSource;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

In the meantime, the CSV data from the local *.csv file type can also be connected to the pivot table via the file uploader option. Here, the resulting string after uploading the file needs to be converted to string array that can be assigned to the [dataSource](#) property under [dataSourceSettings](#). The following code example illustrates the same.

`javascript

```

import { Component, OnInit } from '@angular/core';
import { IDataOptions } from '@syncfusion/ej2-angular-pivotview';
import { Uploader } from '@syncfusion/ej2-inputs';

@Component({
    selector: 'app-container',
    template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>
})
export class AppComponent implements OnInit {
    public dataSourceSettings: IDataOptions;
    public uploadObj: Uploader;
    ngOnInit(): void {
        // Step 1: Initiate the file uploader
    }
}

```

```

this.uploadObj: Uploader = new Uploader({
});
this.uploadObj.appendTo('#fileupload');
// Step 2: Add the event listener which fires when the *.CSV file is uploaded.
let input = document.querySelector('input[type="file"]');
input.addEventListener('change', function (e: Event) {
// Step 3: Initiate the file reader
let reader: FileReader = new FileReader();
reader.onload = function () {
// Step 4: Getting the string output which is to be converted as string[][]
let result: string[][] = (reader.result as string).split('\n').map(function (line) {
return line.split(',');
});
this.dataSourceSettings = {
// Step 5: The string[][] result to be bound as data source
dataSource: result,
type: 'CSV',
// Step 6: The appropriate report needs to be provided here.
};
};
reader.readAsText((input as any).files[0]);
}
}
}
,

```

Binding CSV data via remote

In-order to bind remote CSV data, mention the endpoint [URL](#) under [dataSourceSettings](#) property. The [URL](#) property supports both direct downloadable file (*.csv) and web service URL.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { DataManager } from '@syncfusion/ej2-data';
@Component({

```



```

imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
))
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public data?: DataManager;
    public width?: string;
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            url: 'https://bi.syncfusion.com/productservice/api/sales',
            type: 'CSV',
            expandAll: false,
            enableSorting: true,
            // tslint:disable-next-line:max-line-length
            formatSettings: [{ name: 'Total Cost', format: 'C0' }, { name: 'Total
Revenue', format: 'C0' }, { name: 'Total Profit', format: 'C0' }],
            drilledMembers: [{ name: 'Item Type', items: ['Baby Food' ]}],
            rows: [
                { name: 'Region' },
                { name: 'Country' }
            ],
            columns: [
                { name: 'Item Type' },
                { name: 'Sales Channel' }
            ],
            values: [
                { name: 'Total Cost' },
                { name: 'Total Revenue' },
                { name: 'Total Profit' }
            ],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Remote Data Binding

To interact with remote data source, provide the endpoint `url` within `DataManager` along with appropriate `adaptor`. By default, `DataManager` uses `ODataAdaptor` for remote data-binding.

Binding with OData services

OData is a standardized protocol for creating and consuming data. User can retrieve data from OData service using the **DataManager**. Refer to the following code example for remote data binding using OData service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { DataManager, ODataAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public data?: DataManager;
  public width?: string;
  ngOnInit(): void {
    this.data = new DataManager({
      url:
'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders/',
      adaptor: new ODataAdaptor,
      crossDomain: true
    }).executeQuery(new Query().take(8)).then((e: ReturnOption) => {
      this.dataSourceSettings = {
        dataSource: this.data,
        expandAll: true,
        filters: [],
        columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
        rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
        values: [{ name: 'Freight' }]
      }
    }) as any;
    this.width = '100%';
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Binding with OData V4 services

The OData V4 is an improved version of OData protocols, and the **DataManager** can be used to 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.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { DataManager, ODataV4Adaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public data?: DataManager;
  public width?: string;
  ngOnInit(): void {
    this.data = new DataManager({
      adaptor: new ODataV4Adaptor,
      url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?$top=7',
      crossDomain: true
    });
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: this.data,
      expandAll: true,
      filters: [],
      columns: [{ name: 'OrderDate' }, { name: 'ShipCity' }],
      rows: [{ name: 'OrderID' }, { name: 'CustomerID' }],
      values: [{ name: 'Freight' }]
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Web API

User can use [WebApiAdaptor](#) to bind pivot table with Web API created using OData endpoint.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public data?: DataManager;
  public width?: string;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://bi.syncfusion.com/northwindservice/api/orders',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: this.data,
      expandAll: true,
      filters: [],
      columns: [{ name: 'ProductName', caption: 'Product Name' }],
      rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
      formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
      values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Querying in Data Manager

By default, the data manager retrieves all the data from the provider which is mapped in it. The data from the provider can be filtered, sorted, paged, etc. by setting the own query in `defaultQuery` property in the data manager instance.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { DataManager, WebApiAdaptor, Query, ReturnOption, ODataAdaptor }
from '@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public data?: DataManager;
  public width?: string;
  ngOnInit(): void {
    this.data = new DataManager({
      url:
'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Orders',
      adaptor: new ODataAdaptor(),
      crossDomain: true
    });
    this.data.defaultQuery = new Query().take(2);
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: this.data,
      expandAll: false,
      columns: [{ name: 'CustomerID', caption: 'Customer ID' }],
      rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
      values: [{ name: 'Freight' }]
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Mapping

One can define field information like alias name (caption), data type, aggregation type, show and hide subtotals etc. using the [fieldMapping](#) property under [dataSourceSettings](#). The available options are,

- [name](#) - It is to specify the appropriate field name.
- [caption](#) - It is to set the alias name (caption) to the specific field. Instead of actual field name, the alias name (caption) will be set in the UI of the pivot table.
- [type](#) - It is to display values in the pivot table with appropriate aggregation such as sum, product, count, average, minimum, maximum, etc. Its default value is **sum**. This option is applicable only for relational data source.
- [axis](#) - It will help to display the field in specified axis such as row/column/value/filter axis of the pivot table.
- [showNoDataItems](#) - It is to show all the members of a specific field to the pivot table, even if there are no data in the intersection of the row and column. The default value is **false**. This option is applicable only for relational data source.
- [baseField](#) - For the aggregate types like "DifferenceFrom" or "PercentageOfDifferenceFrom" or "PercentageOfParentTotal", selective field is assigned for comparison via this property.
- [baseItem](#) - For the aggregate types like "DifferenceFrom" or "PercentageOfDifferenceFrom" or "PercentageOfParentTotal", selective member in a field is assigned for comparison via this property.
- [expandAll](#) - It is to expand or collapse all headers of a specific field in row and column axes of the pivot table. The default value is **false**.
- [showSubTotals](#) - It is to show or hide sub-totals of a specific field in row and column axis of the pivot table. The default value is **true**.
- [isNamedSet](#) - It is to set whether the specified field is named set or not. In general, the named set is a set of dimension members or a set expression (MDX query) to be created as a dimension in the SSAS OLAP cube itself. The default value is **false** and this option is applicable only for OLAP data source.
- [isCalculatedField](#) - It is to set whether the specified field is a calculated field or not. In general, a calculated field is created from the bound data source or using simple formula with basic arithmetic operators in the pivot table. The default value is **false** and this option is applicable only for OLAP data source.
- [showFilterIcon](#) - It is to show or hide the filter icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This filter icon is used to filter the members of a specified field at runtime in the pivot table. The default value is **true**.
- [showSortIcon](#) - It is to show or hide the sort icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This sort icon is used to order members of a specified field either in ascending or descending at runtime. The default value is **true**.
- [showRemoveIcon](#) - It is to show or hide the remove icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This remove icon is used to remove the specified field during runtime. The default value is **true**.

- [showValueTypeIcon](#) - It is to show or hide the value type icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This value type icon helps to select the appropriate aggregation type to specified value field at runtime. The default value is **true**.
- [showEditIcon](#) - It is to show or hide the edit icon of a specific field which will be displayed on the button of the grouping bar and field list UI. This edit icon is used to modify caption, formula, and format of a specified calculated field at runtime. The default value is **true**.
- [allowDragAndDrop](#) - It is to restrict specific field's button from being dragged on runtime in the grouping bar and field list UI. This will prevent from altering the current report. The default value is **true**.
- [dataType](#) - It is to specify the type of the field like 'string', 'number', 'datetime', 'date', and 'boolean'.
- [groupName](#) - It is to display fields in the field list UI by grouping them under the desired folder name.

The main purpose of these mapping options is to configure each field that is not part of the initial pivot report. Even if any field that is part of this mapping is defined here, the value set in the initial report will have the highest preceding.

This option is applicable only for relational data source.

In the below code sample, visibility of the field button icons are configured.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService,
FieldListService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService, FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showGroupingBar='true'
showFieldList='true' width=width></ejs-pivotview>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      allowLabelFilter: true,
```

```

        allowValueFilter: true,
        columns: [{ name: 'Year', caption: 'Production Year' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        rows: [{ name: 'Country' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        fieldMapping: [
            { name: 'Quarter', showSortIcon: false },
            { name: 'Products', showFilterIcon: false, showRemoveIcon: false
        },
            { name: 'Amount', showValueTypeIcon: false, caption: 'Sold
Amount' },
        ]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Values in row axis

By default, the value fields are plotted in column axis. To plot those fields in row axis, use the [valueAxis](#) property by setting its value as **row**. By default, it holds the value **column**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  public height?: number;
  ngOnInit(): void {
    this.width = '100%';
  }
}

```



```

        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            valueAxis: 'row'
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Values at different positions

By default, the value fields are placed at the end of the row or column axis. To place those value fields in different positions, use the [valueIndex](#) property and set the value to an appropriate index position. Its default value is -1, which denotes the last position. The [valueIndex](#) property is dependent on the [valueAxis](#) property.

This support is only available for relational data sources. Also, enable the [showValuesButton](#) property in the grouping bar and field list UI to **true** to re-arrange the values fields at different positions via user interaction.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
}

```

```

public width?: string;
public height?: number;
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: [],
        valueIndex: 1
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show 'no data' items

By default, the pivot table only shows the field item if it has data in its row or column combination. To show all items that do not have data in row and column combination in the pivot table, use the [showNoDataItems](#) property by settings its value to **true** for the desired fields. In the following code sample, rows of the "Country" and "State" fields do not have data in all combination with "Date" column field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { noData } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
}

```

```

public width?: string;
public height?: number;
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: noData as IDataset[],
        expandAll: true,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        columns: [{ name: 'Date', showNoDataItems: true }],
        values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country', showNoDataItems: true }, { name: 'State',
showNoDataItems: true }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show value headers always

To show value header always in pivot table, even if it holds a single value, use the [alwaysShowValueHeader](#) property by settings its value as **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  public height?: number;
  ngOnInit(): void {

```

```

        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            alwaysShowValueHeader: true
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize empty value cells

User can show custom string in empty value cells using the [emptyCellsTextContent](#) property in [dataSourceSettings](#) of the pivot table. Since the property is of string data type, user can fill empty value cells with any value like "0", "-", "*", "(blank)", etc. Its common for all value fields and can be configured through code behind.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview';
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { noData } from './datasource';
@Component({
    imports: [
        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    public height?: number;
    ngOnInit(): void {

```

```

        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: noData as IDataset[],
            expandAll: true,
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            columns: [{ name: 'Date', showNoDataItems: true}],
            values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country'}, { name: 'State'}],
            filters: [],
            emptyCellsTextContent: '**'
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Event

Load

The event [load](#) fires before initiate rendering of pivot table. It holds following parameters `dataSourceSettings`, `FieldsType` and `PivotView`. In this event user can customize data source settings before initiating pivot table render module.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, LoadEventArgs } from '@syncfusion/ej2-
angular-pivotview';
import { noData } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'
(load)='load($event)'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    public height?: number;
    load(args: LoadEventArgs | any) {

```

```

        args.dataSourceSettings.emptyCellsTextContent = "###";
        args.dataSourceSettings.columns[0].caption = "Full Year";
        args.dataSourceSettings.expandAll = false;
    }
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: noData as IDataset[],
            expandAll: true,
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            columns: [{ name: 'Date', showNoDataItems: true }],
            values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country'}, { name: 'State'}],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

EnginePopulated

The event [enginePopulated](#) is triggered after engine is populated. It has following parameters - `dataSourceSettings`, `pivotFieldList` and `pivotValues`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, EnginePopulatedEventArgs, IDataset } from
'@syncfusion/ej2-angular-pivotview';
import { noData } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'
(enginePopulated)='enginePopulated($event)'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;

```

```

public height?: number;
enginePopulated(args: EnginePopulatedEventArgs) {
    //trigger after engine is populated
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: noData as IDataset[],
        expandAll: true,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        columns: [{ name: 'Date', showNoDataItems: true}],
        values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country'}, { name: 'State'}],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

EnginePopulating

The event [enginePopulating](#) triggers before the pivot engine starts to populate and allows to customize the pivot datasource settings. It has following parameter `dataSourceSettings`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, EnginePopulatingEventArgs, IDataset } from
'@syncfusion/ej2-angular-pivotview';
import { noData } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'
(enginePopulating)=enginePopulating($event) '></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;

```

```

public height?: number;
enginePopulating(args: EnginePopulatingEventArgs | any) {
    //trigger before engine starts to populate
    args.dataSourceSettings.columns[0].caption = 'Full Year';
    args.dataSourceSettings.emptyCellsTextContent = '###'
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: noData as IDataset[],
        expandAll: true,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        columns: [{ name: 'Date', showNoDataItems: true}],
        values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country'}, { name: 'State'}],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Aggregation](#)
- [Show/Hide Totals](#)
- [Customize number, date, and time values](#)
- [Server Side Engine \(Optional\)](#)

Connecting to data source

MySQL in EJ2 Angular Pivotview Component

This section describes how to retrieve data from a MySQL database using [MySqlClient](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch MySQL data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name

MyWebService

Location

C:\Users\username\source\repos

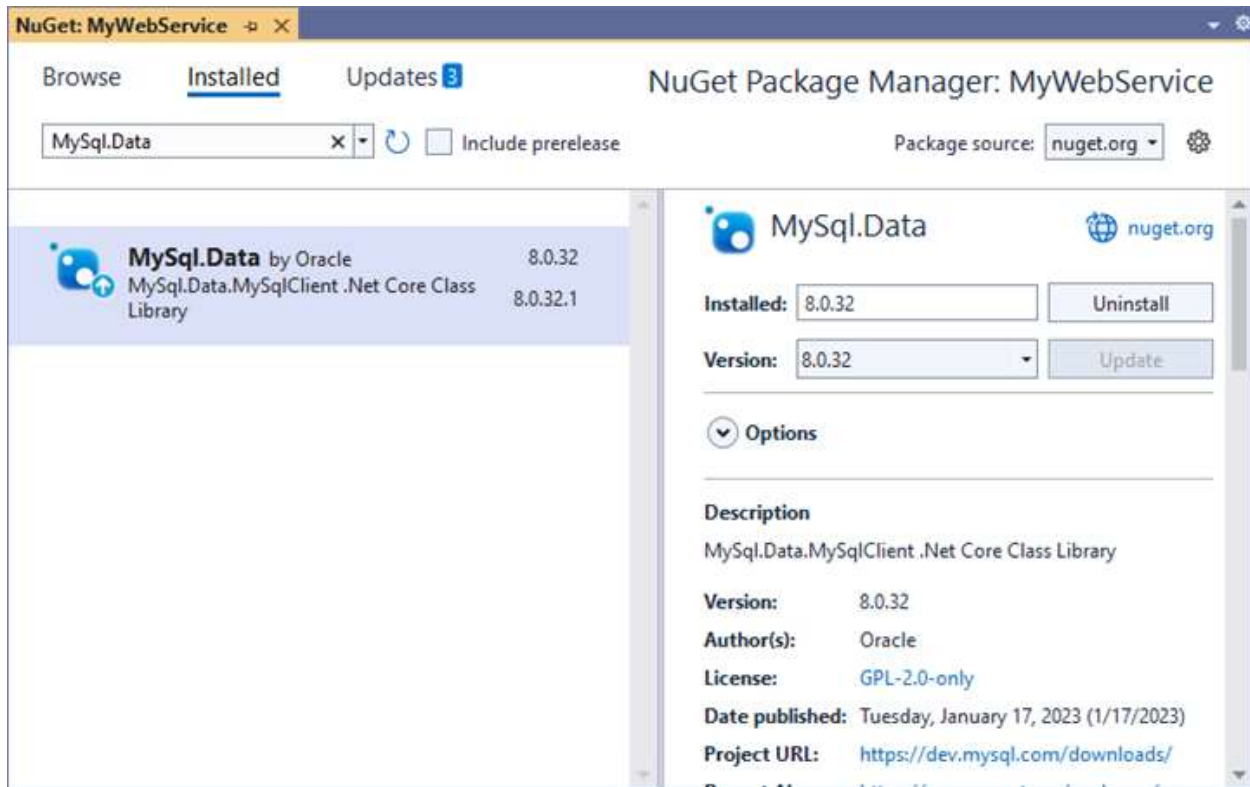
Solution name ⓘ

MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a MySQL Server using the **MySqlClient** in our application, we need to install the [MySql.Data](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **MySql.Data** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **MySQLConnection** helps to connect the MySQL database. Next, using **MySQLCommand** and **MySQLDataAdapter** you can process the desired query string and retrieve data from the MySQL database. The **Fill** method of the **MySQLDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using MySQL.Data.MySqlClient;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        public dynamic GetMySQLResult()
        {
```

```
// Replace with your own connection string.
MySQLConnection connection = new MySqlConnection("<Enter your valid connection string here>");
connection.Open();
MySQLCommand command = new MySQLCommand("SELECT * FROM orders", connection);
MySQLDataAdapter dataAdapter = new MySQLDataAdapter(command);
DataTable dataTable = new DataTable();
dataAdapter.Fill(dataTable);
connection.Close();
return dataTable;
}
}
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **GetMySQLResult** method is used to retrieve the MySQL data as a **DataTable**, which is then serialized into JSON string using **JsonConvert.SerializeObject()**.

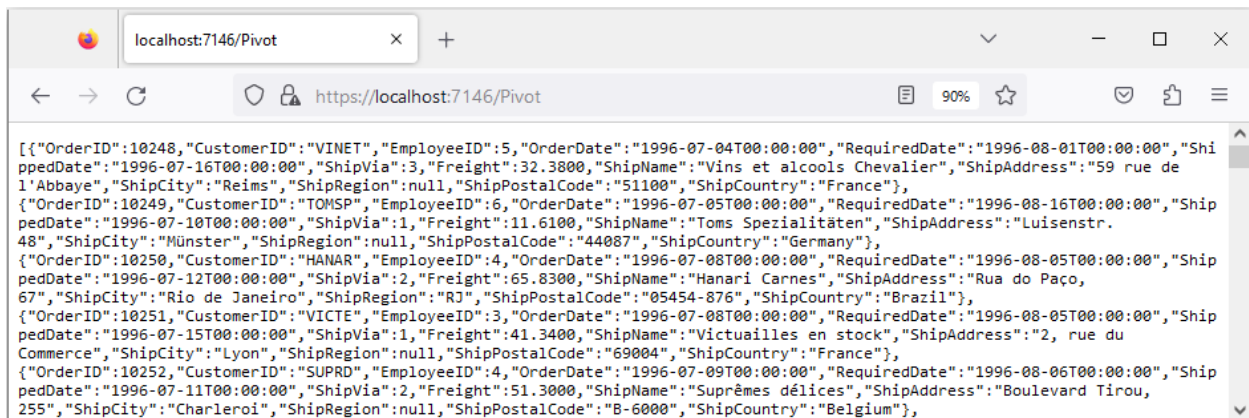
```
`csharp
using Microsoft.AspNetCore.Mvc;
using MySQL.Data.MySqlClient;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetMySQLResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(GetMySQLResult());
        }
        public dynamic GetMySQLResult()
        {

```

```
// Replace with your own connection string.
MySQLConnection connection = new MySqlConnection("<Enter your valid connection string here>");
connection.Open();
MySQLCommand command = new MySqlCommand("SELECT * FROM orders", connection);
MySQLDataAdapter dataAdapter = new MySQLDataAdapter(command);
DataTable dataTable = new DataTable();
dataAdapter.Fill(dataTable);
connection.Close();
return dataTable;
}
}
}
```

6. Run the application and it will be hosted within the URL <https://localhost:7146>.

7. Finally, the retrieved data from MySQL database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:7146/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a MySQL database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).

2. Map the hosted Web API's URL link <https://localhost:7146/Pivot> to the Pivot Table in **app.component.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
```

```
// specifies the template string for the pivot table component
template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'></ejs-pivotview>,
providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:7146/pivot'
    }
    //Other codes here...
  }
}
```

3. Frame and set the report based on the data retrieved from the MySQL database.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:7146/Pivot',
      columns: [{ name: 'ShipName' }],
    }
  }
}
```

```

values: [{ name: 'Freight', caption: 'Sum of Freight' }],
rows: [{ name: 'ShipCity' }],
};
}
}
,

```

When you run the sample, the resulting pivot table will look like this:

	Alfred's Futterkist	Alfreds Futterkist	Ana Trujillo Empa	Antonio Moreno	Around the Horn	B's
Aachen						
Albuquerque						
Anchorage						
Barcelona						
Barquisimeto						
Bergamo						
Berlin	196.12000000...	29.46				
Bern						

Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a MySQL database and bind to the Pivot Table in [this](#) GitHub repository.

Microsoft SQL Server in EJ2 Angular Pivotview Component

This section describes how to retrieve data from SQL Server database using [Microsoft SqlClient](#) and bind it to the Pivot Table via a Web API controller.

Steps to connect the SQL Server database via Web API application

1. Download the ASP.NET Core Web Application from [this](#) GitHub repository.
2. The application named as **PivotController** (server-side) that is downloaded from the above GitHub repository includes the following files.

- **PivotController.cs** file under **Controllers** folder – This helps to do data communication with Pivot Table.
- **Database1.mdf** file under **App_Data** folder – This MDF (Master Database File) file contains example data.

3. In the Web API controller (aka, PivotController), **SqlConnection** helps to connect the SQL database (that is, Database1.mdf). Next, using **SqlCommand** and **SqlDataAdapter** you can process the desired SQL query string and retrieve data from the database. The **Fill** method of the DataAdapter is used to populate the SQL data into a **DataTable** as shown in the following code snippet.

```

`csharp
using Microsoft.AspNetCore.Mvc;

```

```

using System.Data;
using System.Data.SqlClient;
namespace PivotController.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        private static DataTable FetchSQLResult()
        {
            string conSTR = @"<Enter your valid connection string here>";
            string xquery = "SELECT * FROM table1";
            SqlConnection sqlConnection = new(conSTR);
            sqlConnection.Open();
            SqlCommand cmd = new(xquery, sqlConnection);
            SqlDataAdapter dataAdapter = new(cmd);
            DataTable dataTable = new();
            dataAdapter.Fill(dataTable);
            return dataTable;
        }
    }
}

```

4. In the **Get()** method of the **PivotController.cs** file, the **FetchSQLResult** method is used to retrieve the SQL data as a **DataTable**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using System.Data.SqlClient;
namespace PivotController.Controllers
{
    [ApiController]

```

```
[Route("[controller]")]
public class PivotController : ControllerBase
{
    [HttpGet(Name = "GetSQLResult")]
    public object Get()
    {
        return JsonConvert.SerializeObject(FetchSQLResult());
    }
    private static DataTable FetchSQLResult()
    {
        string conSTR = @"<Enter your valid connection string here>";
        string xquery = "SELECT * FROM table1";
        SqlConnection sqlConnection = new(conSTR);
        sqlConnection.Open();
        SqlCommand cmd = new(xquery, sqlConnection);
        SqlDataAdapter dataAdapter = new(cmd);
        DataTable dataTable = new();
        dataAdapter.Fill(dataTable);
        return dataTable;
    }
}
```

5. Run the web application (aka, PivotController) and it will be hosted within the URL <https://localhost:7139>.

6. Finally, the retrieved data from SQL Server which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:7139/pivot>, as shown in the browser page below.



Connecting the Pivot Table to the hosted Web API URL

1. Download the Angular Pivot Table sample from [this](#) GitHub repository.

2. Next, map the hosted Web API's URL link `https://localhost:7139/pivot` to the Pivot Table component in `app.ts` by using the `url` property under `dataSourceSettings`.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: '<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>',
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[] | undefined;
  public dataSourceSettings: IDataOptions | undefined;

  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:7139/pivot'
    }
    //Other codes here...
  }
}
```

```

}
,

```

3. Frame and set the report based on the data retrieved from the SQL database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[] | undefined;
  public dataSourceSettings: IDataOptions | undefined;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:7139/pivot',
      enableSorting: true,
      expandAll: false,
      columns: [{ name: 'Product' }],
      values: [{ name: 'Quantity' }, { name: 'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'State' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}
,

```

4. Run the sample to get the following result.

	Large Vehicle		Small Vehicle		Grand Total	
	Quantity	Amount	Quantity	Amount	Quantity	Am
▶ Australia	16	24000	12	2000	28	
▶ Canada			420	1649000	420	
▶ Jamaica	400	300120			400	
▶ Trinidad			328	453040	328	
Grand Total	416	324120	760	2104040	1176	

The sample for connecting the Pivot Table to a SQL Server database via an ASP.NET Web application can be found in [this](#) GitHub repository.

PostgreSQL in EJ2 Angular Pivotview Component

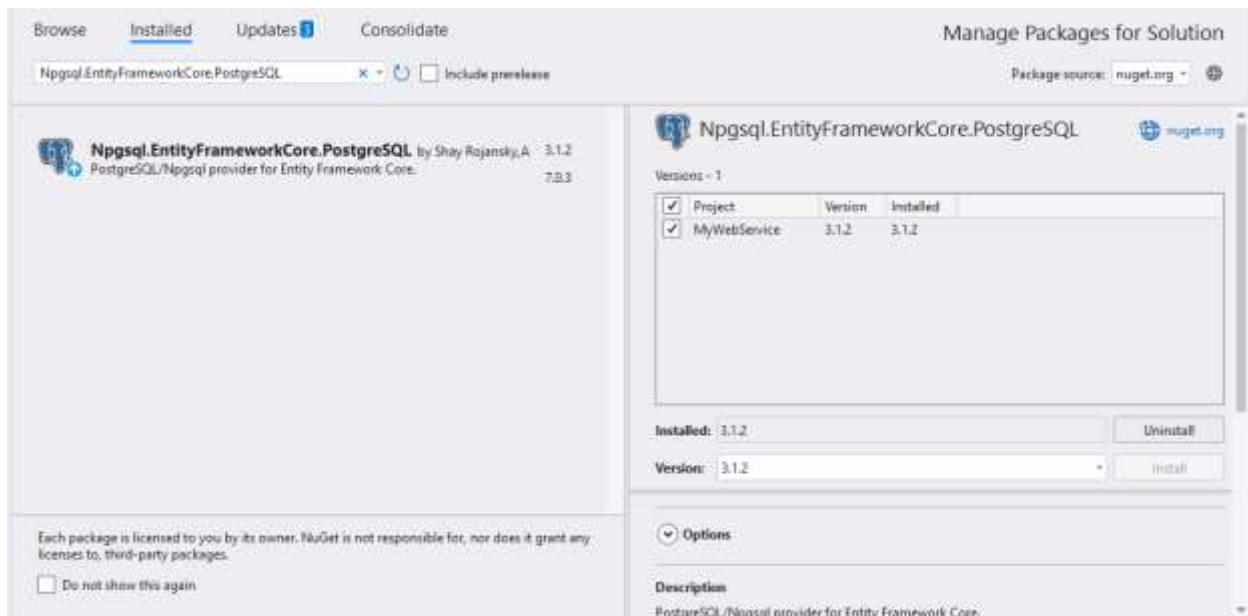
This section describes how to consume data from PostgreSQL database using [Microsoft Npgsql](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch PostgreSQL data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

The screenshot shows the 'Configure your new project' window in Visual Studio. The 'ASP.NET Core Web App' template is selected. The project name is 'MyWebService', the location is 'C:\Users\username\sources\repos', and the solution name is 'MyWebService'. The checkbox 'Place solution and project in the same directory' is unchecked. 'Back' and 'Next' buttons are at the bottom right.

2. To connect a PostgreSQL Server using the **Npgsql** in our application, we need to install the [Npgsql.EntityFrameworkCore.PostgreSQL](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Npgsql.EntityFrameworkCore.PostgreSQL** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **NpgsqlConnection** helps to connect the PostgreSQL database. Next, using **NpgsqlCommand** and **NpgsqlDataAdapter** you can process the desired PostgreSQL query string and retrieve data from the database. The **Fill** method of the **NpgsqlDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using Npgsql;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        public dynamic GetPostgreSQLResult()
        {
```

```
// Replace with your own connection string.
NpgsqlConnection connection = new NpgsqlConnection("<Enter your valid connection string here>");
connection.Open();
NpgsqlCommand cmd = new NpgsqlCommand("SELECT * FROM tablename", connection);
NpgsqlDataAdapter da = new NpgsqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
connection.Close();
return dt;
}
}
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **GetPostgreSQLResult** method is used to retrieve the PostgreSQL data as a **DataTable**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

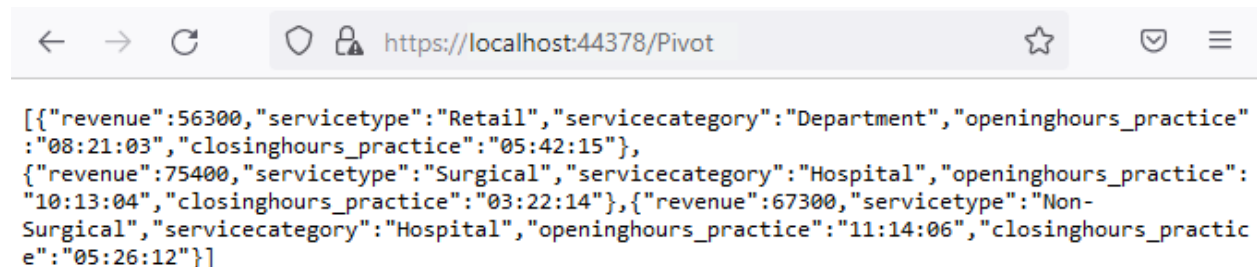
```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Data;
using Npgsql;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetPostgreSQLResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(GetPostgreSQLResult());
        }
        public dynamic GetPostgreSQLResult()
        {

```

```
// Replace with your own connection string.
NpgsqlConnection connection = new NpgsqlConnection("<Enter your valid connection string here>");
connection.Open();
NpgsqlCommand cmd = new NpgsqlCommand("SELECT * FROM tablename", connection);
NpgsqlDataAdapter da = new NpgsqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
connection.Close();
return dt;
}
}
}
```

6. Run the application and it will be hosted within the URL <https://localhost:44378/>.

7. Finally, the retrieved data from PostgreSQL database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44378/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a PostgreSQL database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).

2. Map the hosted Web API's URL link <https://localhost:44378/Pivot> to the Pivot Table component in **app.component.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>
```

```

providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44378/pivot'
      //Other codes here...
    };
  }
}
`

```

3. Frame and set the report based on the data retrieved from the PostgreSQL database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44378/Pivot',
      enableSorting: true,
      columns: [{ name: 'openinghourspractice' }, { name: 'closinghourspractice' } ],
      values: [{ name: 'revenue' } ],
    };
  }
}
`

```

```
rows: [{ name: 'servicetype' }, { name: 'servicecategory' } ]
};
}
}
,
```

When you run the sample, the resulting pivot table will look like this:

	▶ 08:21:03	▶ 10:13:04	▶ 11:14:06	Grand Total
▶ Non-Surgical			67300	67300
▶ Retail	56300			56300
▶ Surgical		75400		75400
Grand Total	56300	75400	67300	199000

Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a PostgreSQL database and bind to the Pivot Table in [this](#) GitHub repository.

Oracle in EJ2 Angular Pivotview Component

This section describes how to retrieve data from Oracle database using [Oracle Managed Data Access](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Oracle data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Oracle Server using the **Oracle.ManagedDataAccess.Client** in our application, we need to install the [Oracle.ManagedDataAccess.Core](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Oracle.ManagedDataAccess.Core** and install it.

Manage Packages for Solution

Package source: nuget.org

Oracle.ManagedDataAccess.Core

Project	Version	Installed
MyWebService	3.21.80	3.21.80

Installed: 3.21.80 Uninstall

Version: 3.21.80 Install

Options

Description

Oracle Data Provider for .NET (ODP.NET) Core is an ADO.NET driver that provides fast data access from Microsoft .NET Core clients to Oracle databases. ODP.NET Core consists of a single 100% managed code dynamic-link library.

3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **OracleConnection** helps to connect the Oracle database. Next, using **OracleCommand** and **OracleDataAdapter** you can process the desired Oracle query string and retrieve data from the database. The **Fill** method of the **OracleDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Oracle.ManagedDataAccess.Client;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        private static DataTable FetchOracleResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            OracleConnection oracleConnection = new OracleConnection(connectionString);
            oracleConnection.Open();
            OracleCommand command = new OracleCommand("SELECT * FROM EMPLOYEES", oracleConnection);
            OracleDataAdapter dataAdapter = new OracleDataAdapter(command);
            DataTable dataTable = new DataTable();
            dataAdapter.Fill(dataTable);
            oracleConnection.Close();
            return dataTable;
        }
    }
}
```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchOracleResult()** method is used to retrieve the Oracle data, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Oracle.ManagedDataAccess.Client;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetOracleResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchOracleResult());
        }
        private static DataTable FetchOracleResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            OracleConnection oracleConnection = new OracleConnection(connectionString);
            oracleConnection.Open();
            OracleCommand command = new OracleCommand("SELECT * FROM EMPLOYEES", oracleConnection);
            OracleDataAdapter dataAdapter = new OracleDataAdapter(command);
            DataTable dataTable = new DataTable();
            dataAdapter.Fill(dataTable);
            oracleConnection.Close();
            return dataTable;
        }
    }
}
```

6. Run the application and it will be hosted within the URL <https://localhost:44346/>.

7. Finally, the retrieved data from Oracle database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44346/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a Oracle database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).
2. Map the hosted Web API's URL link <https://localhost:44346/Pivot> to the Pivot Table component in **app.component.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44346/pivot'
      //Other codes here...
    };
  }
}
```

```

}
}
`

```

3. Frame and set the report based on the data retrieved from the Oracle database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44346/pivot',
      enableSorting: true,
      expandAll: false,
      columns: [
        { name: 'DEPARTMENT_ID', caption: 'Department ID' },
        { name: 'EMPLOYEE_NAME', caption: 'Employee Name' },
      ],
      rows: [
        { name: 'JOB', caption: 'Job' },
        { name: 'SALARY', caption: 'Salary' }
      ],
      values: [
        { name: 'EMPLOYEE_ID', caption: 'Employee ID' },
        { name: 'CC_EMPLOYEES', caption: 'Employees' },
      ],
    };
  }
}

```

```
{ name: 'CCTAXPERCENTAGE', caption: 'Percentage' },
],
filters: []
};
}
}
,
```

When you run the sample, the resulting pivot table will look like this:

	▶ 10	▶ 20	▶ 30	Grand Total
▶ ANALYST		15690		15690
▶ CLERK	7934	15245	7900	31079
▶ MANAGER	7782	7566	7698	23046
▶ PRESIDENT	7839			7839
▶ SALESMAN			30518	30518
Grand Total	23555	38501	46116	108172

Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a Oracle database and bind to the Pivot Table in [this](#) GitHub repository.

MongoDB in EJ2 Angular Pivotview Component

This section describes how to consume data from MongoDB database using [MongoDB Driver](#) and [MongoDB Bson](#) libraries and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch MongoDB data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a MongoDB Server using the **MongoDB.Driver** and **MongoDB.Bson** in our application, we need to install the [MongoDB.Driver](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **MongoDB.Driver** and install it.

Manage Packages for Solution

Browse Installed Updates 2 Consolidate

MongoDB.Driver Include prerelease Package source: nuget.org

MongoDB.Driver by MongoDB Inc. 2.19.0
Official .NET driver for MongoDB.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.
☐ Do not show this again

MongoDB.Driver nuget.org

Versions - 1

Project	Version	Installed
MyWebService	2.19.0	2.19.0

Installed: 2.19.0 Uninstall

Version: Latest stable 2.19.0 Install

3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **MongoClient** helps to connect the MongoDB database. Next, using the **GetDatabase** and **GetCollection** methods, you can retrieve data from the database. The **Find** method of the **IMongoDatabase** is used to populate the retrieved data into a **List**, as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using MongoDB.Driver;
using MongoDB.Bson;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        private static List<ProductDetails> FetchMongoDbResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            MongoClient client = new MongoClient(connectionString);
            IMongoDatabase database = client.GetDatabase("sample_training");
            var collection = database.GetCollection<ProductDetails>("ProductDetails");
            return collection.Find(new BsonDocument()).ToList();
        }

        public class ProductDetails
        {
            public ObjectId Id { get; set; }
            public int Sold { get; set; }
            public double Amount { get; set; }
            public string? Country { get; set; }
            public string? Products { get; set; }
            public string? Year { get; set; }
        }
    }
}
```



```

public string? Quarter { get; set; }
}
}
}
`

```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchMongoDbResult()** method is used to retrieve the MongoDB data as a **List**, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```

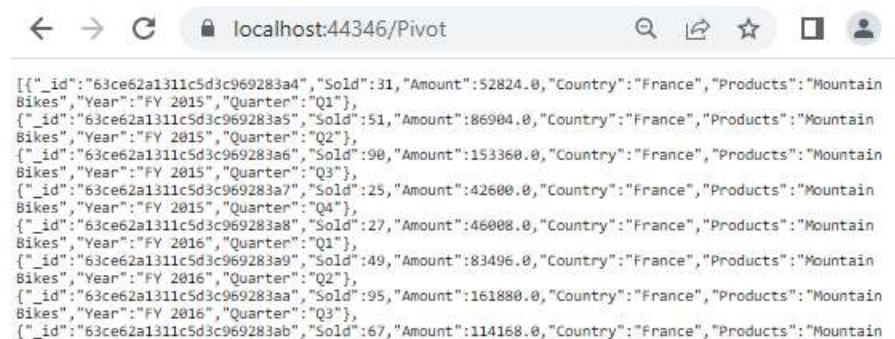
`csharp
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using MongoDB.Bson;
using MongoDB.Driver;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetMongoDbResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchMongoDbResult());
        }
        private static List<ProductDetails> FetchMongoDbResult()
        {
            // Replace with your own connection string.
            string connectionString = "<Enter your valid connection string here>";
            MongoClient client = new MongoClient(connectionString);
            IMongoDatabase database = client.GetDatabase("sample_training");
            var collection = database.GetCollection<ProductDetails>("ProductDetails");
            return collection.Find(new BsonDocument()).ToList();
        }
        public class ProductDetails

```

```
{
  public ObjectId Id { get; set; }
  public int Sold { get; set; }
  public double Amount { get; set; }
  public string? Country { get; set; }
  public string? Products { get; set; }
  public string? Year { get; set; }
  public string? Quarter { get; set; }
}
}
```

6. Run the web application and it will be hosted within the URL <https://localhost:44346/>.

7. Finally, the retrieved data from MongoDB database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44346/Pivot>, as shown in the browser page below.



```
[{"_id": "63ce62a1311c5d3c969283a4", "Sold": 31, "Amount": 52824.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2015", "Quarter": "Q1"}, {"_id": "63ce62a1311c5d3c969283a5", "Sold": 51, "Amount": 86904.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2015", "Quarter": "Q2"}, {"_id": "63ce62a1311c5d3c969283a6", "Sold": 90, "Amount": 153360.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2015", "Quarter": "Q3"}, {"_id": "63ce62a1311c5d3c969283a7", "Sold": 25, "Amount": 42600.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2015", "Quarter": "Q4"}, {"_id": "63ce62a1311c5d3c969283a8", "Sold": 27, "Amount": 46008.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2016", "Quarter": "Q1"}, {"_id": "63ce62a1311c5d3c969283a9", "Sold": 49, "Amount": 83496.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2016", "Quarter": "Q2"}, {"_id": "63ce62a1311c5d3c969283aa", "Sold": 95, "Amount": 161880.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2016", "Quarter": "Q3"}, {"_id": "63ce62a1311c5d3c969283ab", "Sold": 67, "Amount": 114168.0, "Country": "France", "Products": "Mountain Bikes", "Year": "FY 2016", "Quarter": "Q4"}]
```

Connecting the Pivot Table to a MongoDB database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).

2. Map the hosted Web API's URL link <https://localhost:44346/Pivot> to the Pivot Table component in **app.component.ts** by using the [url](#) property under [dataSourceSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
```

```

template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
providers: [FieldListService],
})
export class AppComponent implements OnInit {
public pivotData: IDataset[];
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
url: 'https://localhost:44346/pivot'
//Other codes here...
};
}
}
`

```

3. Frame and set the report based on the data retrieved from the MongoDB database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
selector: 'app-root',
// specifies the template string for the pivot table component
template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
providers: [FieldListService],
})
export class AppComponent implements OnInit {
public pivotData: IDataset[];
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
url: 'https://localhost:44346/Pivot',
enableSorting: true,
columns: [

```

```

{ name: 'Year' }
],
values: [
{ name: 'Sold', caption: "Units Sold"},
{ name: 'Amount', caption: "Sold Amount"}
],
rows: [
{ name: 'Country' },
{ name: 'Products' }
]
];
}
}
,

```

When you run the sample, the resulting pivot table will look like this:

	FY 2015		FY 2016		F
	Units Sold	Sold Amount	Units Sold	Sold Amount	U
> France	729	1160099.5	609	983317	
> Germany	528	845472	667	1067220	
> United Kingdom	782	1263109.5	640	1031630.5	
> United States	682	1085398.5	480	770362	
Grand Total	2721	4354079.5	2396	3852529.5	

Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a MongoDB database and bind to the Pivot Table in [this](#) GitHub repository.

Elasticsearch in EJ2 Angular Pivotview Component

This section describes how to retrieve data from Elasticsearch database using [Nest](#) library and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Elasticsearch data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Elasticsearch Server using the **NEST** in our application, we need to install the [NEST](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **NEST** and install it.

Browse Installed Updates 1 Consolidate Manage Packages for Solution

NEST x Include prerelease Package source: nuget.org

NEST by Elastic and contributors 7.17.5
Strongly typed interface to Elasticsearch. Fluent and classic object initializer mappings of requests and

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.
☐ Do not show this again

Versions - 1	
	Version
<input checked="" type="checkbox"/> Project	7.17.5
<input checked="" type="checkbox"/> MyWebService	7.17.5

Installed: 7.17.5 Uninstall

Version: Latest stable 7.17.5 Install

3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **ElasticClient** helps to connect the Elasticsearch database. Next, using **Search** method you can query your Elasticsearch index and retrieve results from the database.

5. In the **Get()** method of the **PivotController.cs** file, the **FetchElasticsearchData** method is used to retrieve the Elasticsearch data, which is then serialized into JSON using **JsonConvert.SerializeObject()**.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Nest;
using Newtonsoft.Json;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetElasticSearchData")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchElasticsearchData());
        }
        private static object FetchElasticsearchData()
        {
            // Replace with your own connection string.
            var connectionString = "<Enter your valid connection string here>";
            var uri = new Uri(connectionString);
            var connectionSettings = new ConnectionSettings(uri);
            var client = new ElasticClient(connectionSettings);
            var searchResponse = client.Search<object>(s => s
                .Index("product")
                .Size(1000)
            );
            return searchResponse.Documents;
        }
    }
}
```

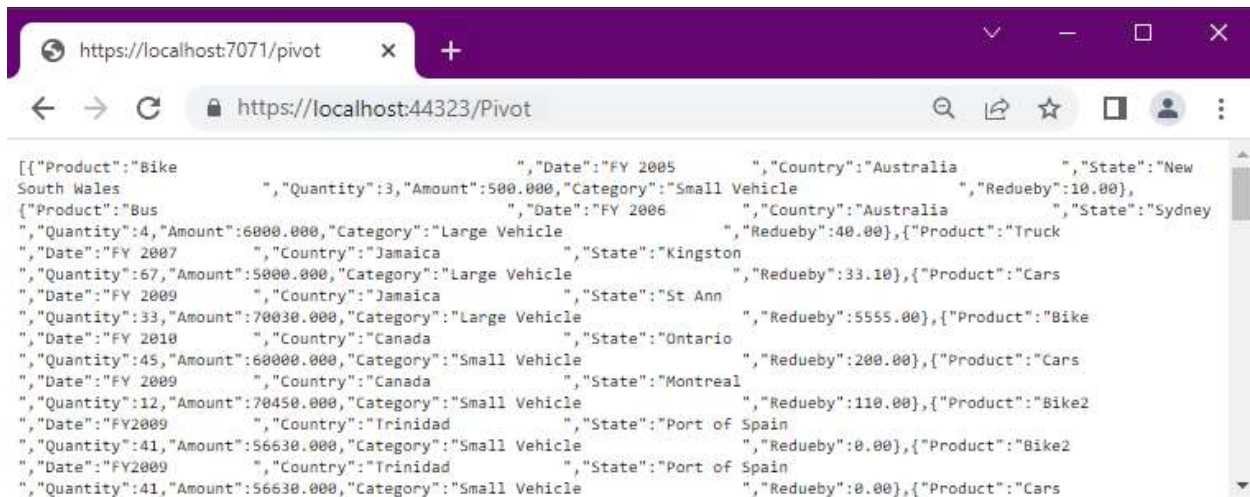
```

}
}
}
`

```

6. Run the web application and it will be hosted within the URL <https://localhost:44323>.

7. Finally, the retrieved data from Elasticsearch database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44323/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to an Elasticsearch database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).

2. Map the hosted Web API's URL link <https://localhost:44323/Pivot> to the Pivot Table component in `app.component.ts` by using the `url` property under `dataSourceSettings`.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[] | undefined;
  public dataSourceSettings: IDataOptions | undefined;

```

```

ngOnInit(): void {
  this.dataSourceSettings = {
    url: 'https://localhost:44323/pivot'
    //Other codes here...
  };
}
}
`

```

3. Frame and set the report based on the data retrieved from the Elasticsearch database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[showFieldList]="true"></ejs-pivotview>`,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[] | undefined;
  public dataSourceSettings: IDataOptions | undefined;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44323/Pivot',
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Product' }],
      values: [{ name: 'Quantity' }, { name: 'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'State' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    }
  }
}

```



```
};
}
}
,
```

When you run the sample, the resulting pivot table will look like this:

	▶ Large Vehicle		▶ Small Vehicle		Grand Total	
	Quantity	Amount	Quantity	Amount	Quantity	Am
▶ Australia	16	24000	12	2000	28	
▶ Canada			420	1649000	420	
▶ Jamaica	400	300120			400	
▶ Trinidad			328	453040	328	
Grand Total	416	324120	760	2104040	1176	

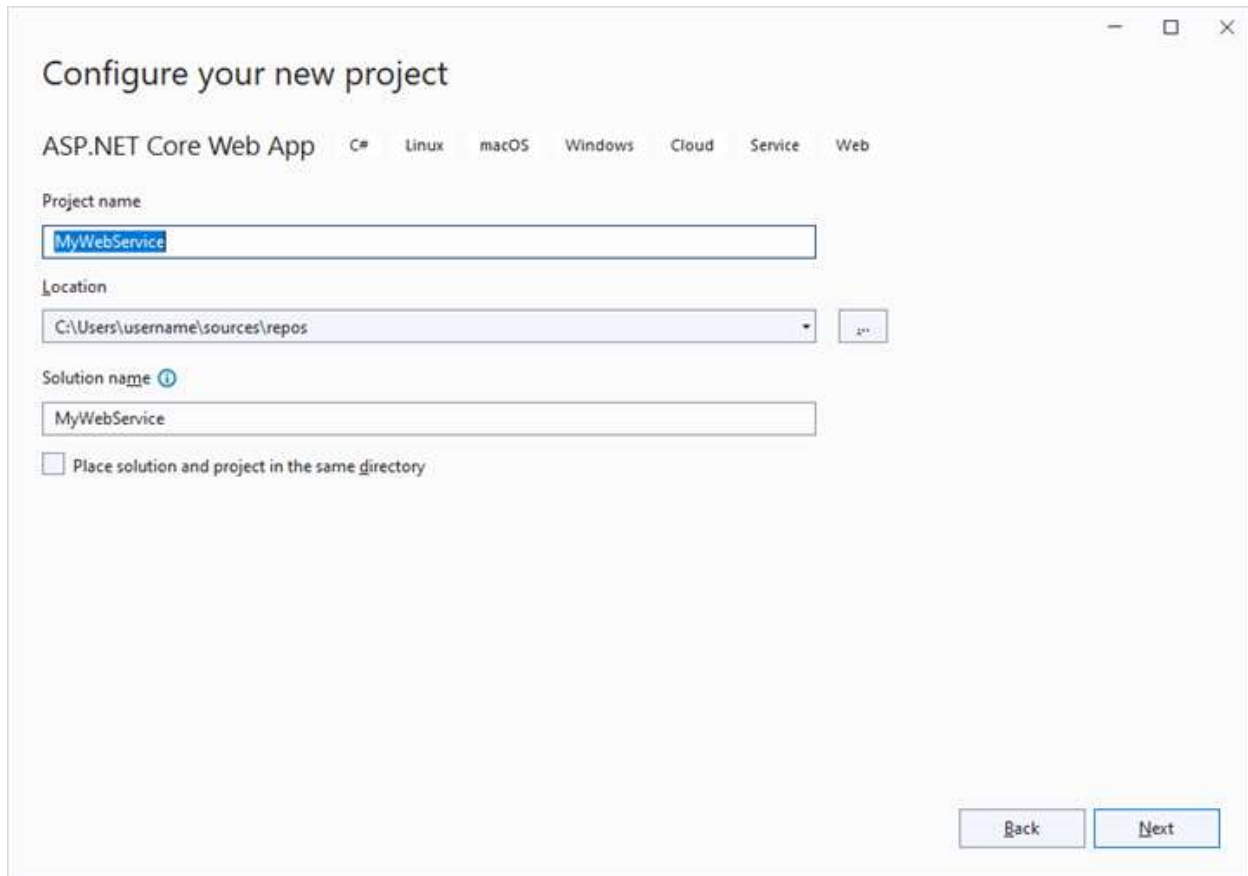
Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a Elasticsearch database and bind to the Pivot Table in [this](#) GitHub repository.

Snowflake in EJ2 Angular Pivotview Component

This section describes how to retrieve data from a Snowflake database using [Snowflake Data](#) and bind it to the Pivot Table via a Web API controller.

Create a Web API service to fetch Snowflake data

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).



Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name
MyWebService

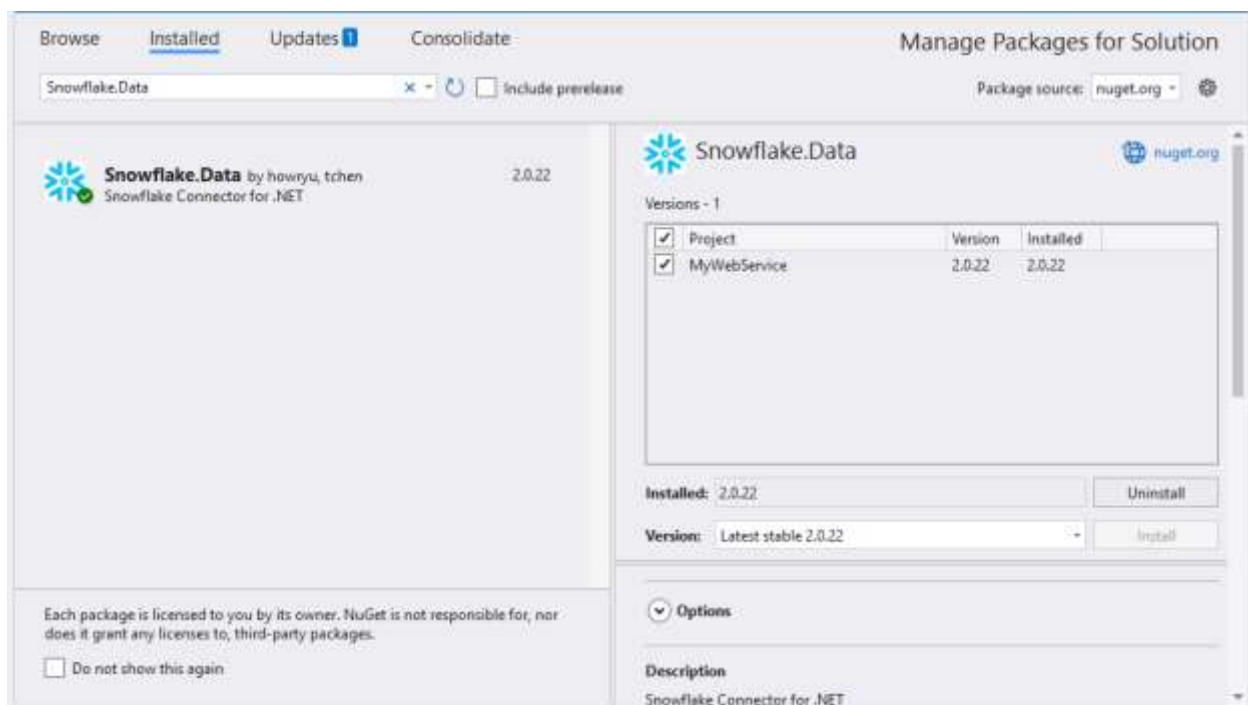
Location
C:\Users\username\source\repos

Solution name
MyWebService

☐ Place solution and project in the same directory

Back Next

2. To connect a Snowflake Server using the **Snowflake.Data.Client** in our application, we need to install the [Snowflake.Data](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Snowflake.Data** and install it.



3. Create a Web API controller (aka, PivotController.cs) file under **Controllers** folder that helps to establish data communication with the Pivot Table.

4. In the Web API controller (aka, PivotController), **SnowflakeDbConnection** helps to connect the Snowflake database. Next, using **SnowflakeDbDataAdapter** you can process the desired Snowflake query string and retrieve data from the database. The **Fill** method of the **SnowflakeDbDataAdapter** is used to populate the retrieved data into a **DataTable** as shown in the following code snippet.

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Snowflake.Data.Client;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetSnowflakeResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchSnowflakeResult());
        }
        public static DataTable FetchSnowflakeResult()
        {
            using (SnowflakeDbConnection snowflakeConnection = new SnowflakeDbConnection())
            {
                // Replace with your own connection string.
                snowflakeConnection.ConnectionString = "<Enter your valid connection string here>";
                snowflakeConnection.Open();
                SnowflakeDbDataAdapter adapter = new SnowflakeDbDataAdapter("select * from CALL_CENTER",
                    snowflakeConnection);
                DataTable dataTable = new DataTable();
                adapter.Fill(dataTable);
                snowflakeConnection.Close();
                return dataTable;
            }
        }
    }
}
```

```

}
}
}
}
`

```

5. In the **Get()** method of the **PivotController.cs** file, the **FetchSnowflakeResult** method is used to retrieve the Snowflake data as a **DataTable**, which is then serialized into JSON string using **JsonConvert.SerializeObject()**.

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Snowflake.Data.Client;
using Newtonsoft.Json;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpGet(Name = "GetSnowflakeResult")]
        public object Get()
        {
            return JsonConvert.SerializeObject(FetchSnowflakeResult());
        }
        public static DataTable FetchSnowflakeResult()
        {
            using (SnowflakeDbConnection snowflakeConnection = new SnowflakeDbConnection())
            {
                // Replace with your own connection string.
                snowflakeConnection.ConnectionString = "<Enter your valid connection string here>";
                snowflakeConnection.Open();
                SnowflakeDbDataAdapter adapter = new SnowflakeDbDataAdapter("select * from CALL_CENTER",
                    snowflakeConnection);
                DataTable dataTable = new DataTable();
            }
        }
    }
}

```

```

adapter.Fill(dataTable);
snowflakeConnection.Close();
return dataTable;
}
}
}
}
,

```

6. Run the application and it will be hosted within the URL <https://localhost:44378/>.

7. Finally, the retrieved data from Snowflake database which is in the form of JSON can be found in the Web API controller available in the URL link <https://localhost:44378/Pivot>, as shown in the browser page below.



Connecting the Pivot Table to a Snowflake database using the Web API service

1. Create a simple Angular Pivot Table by following the “Getting Started” documentation [link](#).

2. Map the hosted Web API's URL link <https://localhost:44378/Pivot> to the Pivot Table component in **app.component.ts** by using the [url](#) property under [dataSourceSettings](#).

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})

```

```

export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44378/Pivot',
      //Other codes here...
    };
  }
}

```

3. Frame and set the report based on the data retrieved from the Snowflake database.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { FieldListService, IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>,
  providers: [FieldListService],
})
export class AppComponent implements OnInit {
  public pivotData: IDataset[];
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      url: 'https://localhost:44378/Pivot',
      enableSorting: true,
      expandAll: false,
      dataSource: [],
      columns: [
        { name: 'CC_COUNTRY', caption: 'Country' }
      ]
    };
  }
}

```

```

],
rows: [
{ name: 'CC_STATE', caption: 'State' },
{ name: 'CC_CITY', caption: 'City' }
],
values: [
{ name: 'CC_COMPANY', caption: 'Company' },
{ name: 'CC_EMPLOYEES', caption: 'Employees' },
{ name: 'CCTAXPERCENTAGE', caption: 'Percentage' },
],
filters: []
};
}
}
,

```

When you run the sample, the resulting pivot table will look like this:

	United States			Grand Total		
	Company	Employees	Percentage	Company	Employees	Percentage
▶ CA	16	413279139	0.4800000000...	16	413279139	0.480000...
▶ CO	5	1160903623	0.18	5	1160903623	
▶ FL	15	1162362540	0.33	15	1162362540	
▶ GA	20	2358562323	0.12	20	2358562323	
▶ IL	8	707852970	0.1699999999...	8	707852970	0.169999'
▶ KS	9	1842349179	0.19	9	1842349179	
▶ LA	6	528468075	0.16	6	528468075	

Explore our Angular Pivot Table sample and ASP.NET Core Web Application to extract data from a Snowflake database and bind to the Pivot Table in [this](#) GitHub repository.

Olap in Angular Pivot Table component

Getting started

This section explain steps to create a simple **Pivot Table** with OLAP data source in an Angular environment.

Dependencies

The following list of dependencies are required to use the pivot table component in your application.

```
`javascript
```

```
|-- @syncfusion/ej2-angular-pivotview
|-- @syncfusion/ej2-pivotview
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-excel-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-pdf-export
|-- @syncfusion/ej2-file-utils
|-- @syncfusion/ej2-compression
|-- @syncfusion/ej2-calendars
|-- @syncfusion/ej2-charts
|-- @syncfusion/ej2-svg-base
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-angular-base
\
```

Setup Angular Environment

You can use [Angular CLI](#) to setup your Angular application. To install Angular CLI use the following command.

```
`bash
npm install -g @angular/cli
\
```

Create an Angular Application

Create a new Angular application using below Angular CLI command.

```
`bash
ng new my-app
\
```


Now, the application is created in the **my-app** demo folder. Run the following command to navigate to the **my-app** demo folder.

```
`bash
cd my-app
`
```

Adding Syncfusion PivotView package

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) registry. To install the pivot table component, use the following command.

```
`bash
npm install @syncfusion/ej2-angular-pivotview --save
`
```

The **--save** will instruct NPM to include the pivot table package inside the **dependencies** section of the **package.json**.

Registering PivotView Module

Import **PivotViewModule** (aka, PivotTable) into an Angular application in **src/app/app.module.ts** file from the package **@syncfusion/ej2-angular-pivotview**. Then declare **PivotViewModule** in imports section of the NgModule as follows.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the PivotViewModule for the pivot table component
import { PivotViewModule } from '@syncfusion/ej2-angular-pivotview';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-pivotview module into NgModule
  imports: [ BrowserModule, PivotViewModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

Adding CSS reference

Next we need to refer the pivot table and its [dependency](#) styles into the **[src/styles.css]** file. Those CSS files are available in **"../node_modules/@syncfusion"** package folders. This can be referenced in **[src/styles.css]** using following code. In this illustration, we have referred **material** theme.

```
`css
```

```

@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-grids/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-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-pivotview/styles/material.css';
`

```

You can also refer other themes like bootstrap, fabric, high-contrast etc. To know about individual component CSS, please refer [here](#).

Add pivot table component

Next we need to add the template in **[src/app/app.component.ts]** file to initialize the pivot table component. Add the Angular pivot table by using `<ejs-pivotview>` selector in **template** section of the **app.component.ts** file.

```

`typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  template: <ejs-pivotview></ejs-pivotview>
})
export class AppComponent {
}
`

```

After initialization, add the the following code in **[src/app/app.component.ts]** file to populate pivot table with a sample OLAP data source. Refer [here](#) to know the more details about OLAP data binding.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { IDataOptions } from '@syncfusion/ej2-angular-pivotview';

@Component({
  selector: 'app-root',

```

```
// specifies the template string for the pivot table component
template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033
    };
  }
}
```

Adding OLAP cube elements to row, column, value and filter axes

Now that pivot table is initialized and assigned with sample OLAP data source, will further move to showcase the component by organizing appropriate [OLAP cube elements](#) in [rows](#), [columns](#), [values](#) and [filters](#) axes.

In [dataSourceSettings](#) property, four major axes [rows](#), [columns](#), [values](#) and [filters](#) plays a vital role in defining and organizing [OLAP cube elements](#) from the bound data source, to render the entire pivot table component in a desired format.

[rows](#) – Collection of [OLAP cube elements](#) (such as Hierarchies, NamedSet, Calculated Members etc.,) that needs to be displayed in row axis of the pivot table.

[columns](#) – Collection of [OLAP cube elements](#) (such as Hierarchies, NamedSet, Calculated Members etc.,) that needs to be displayed in column axis of the pivot table.

[values](#) – Collection of [OLAP cube elements](#) (such as Measures, Calculated Measures) that needs to be displayed as aggregated numeric values in the pivot table.

[filters](#) - Collection of [OLAP cube elements](#) (such as Hierarchies and Calculated Members) that would act as master filter over the data bound in row, column and value axes of the pivot table.

In-order to define each [OLAP cube element](#) in the respective axis, the following basic properties should be set.

- **name**: It allows to set the unique name of the hierarchies, named set, measures, calculated members etc., from the bound OLAP data source. It's casing should match exactly like in the data source and if not set properly, the pivot table will be rendered as empty.
- **caption**: It allows to set the caption, which is the alias name of the unique name that needs to be displayed in the pivot table. If not provided, unique name will be displayed.

In this sample, "Product Categories" is added in column, "Customer Geography" in row, and "Customer Count" and "Internet Sales Amount" in value axes respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
        { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
      ],
    },
  ],
}
```

```

        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
[ '[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]' ],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Applying formatting to a value field

Formatting defines a way in which values should be displayed. For example, format “C” denotes the values should be displayed in currency pattern. To do so, define the [formatSettings](#) property with its [name](#) and [format](#) properties and add it to [formatSettings](#). In this sample, the [name](#) property is set as **[Measures].[Internet Sales Amount]**, a field from value section and its format is set as currency. Likewise, we can set format for other value fields as well and add it to [formatSettings](#).

Only fields from value section, which is in the form of numeric data values are applicable for formatting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
}

```

```

public width?: string;
ngOnInit(): void {
    this.dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ],
        formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Enable grouping bar

The Grouping Bar feature automatically populates [OLAP cube elements](#) from the bound data source and allows end users to drag [OLAP cube elements](#) between different axes such as [rows](#), [columns](#), [values](#) and [filters](#), and change pivot view at runtime. Sorting, filtering and removing of elements is also possible. It

can be enabled by setting the [showGroupingBar](#) property to **true** and by injecting the **GroupingBarService** module as follows.

If the **GroupingBarService** module is not injected, the grouping bar will not be rendered with the pivot table component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService } from
 '@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showGroupingBar='true'></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
        { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
      ],
      filters: [
        { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },

```

```

        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
[ '[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]' ],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Enable pivot field list

The component provides a built-in Field List similar to Microsoft Excel. It allows you to add or remove [OLAP cube elements](#) and also rearrange the [OLAP cube elements](#) between different axes, including [rows](#), [columns](#), [values](#) and [filters](#) along with filter and sort options dynamically at runtime. It can be enabled by setting the [showFieldList](#) property to **true** and by injecting the **FieldListService** module as follows.

If the **FieldListService** module is not injected, the Field List will not be rendered with the pivot table component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {

```



```

public dataSourceSettings?: IDataOptions;
public width?: string;
ngOnInit(): void {
    this.dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Exploring filter axis

The filter axis contains collection of [OLAP cube elements](#) such as hierarchies and calculated members that would act as master filter over the data bound in [rows](#), [columns](#) and [values](#) axes of the pivot table. The [OLAP cube elements](#) along with filter members could be set to filter axis either through report via code behind or by dragging and dropping [OLAP cube elements](#) from other axes to filter axis via grouping bar or field list at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
        { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
      ],
      filters: [
        { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
      ],
      filterSettings: [
        {
          name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],

```

```

        levelCount: 3
    }
    ]
};
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Calculated field

The calculated field allows user to insert or add a new calculated field based on the available [OLAP cube elements](#) from the bound data source. Calculated fields are nothing but customized dimensions or measures that are newly created based on the user-defined expression.

The two types of calculated fields are as follows:

- **Calculated Measure** – Creates a new measure through user-defined expression.
- **Calculated Dimension** – Creates a new dimension through user-defined expression.

It can be customized using the [calculatedFieldsSettings](#) property through code behind. The setting required for calculate field feature at code behind are:

- [name](#): It allows to set the unique name for new calculated field.
- [formula](#): It allows to set the user-defined expression.
- [hierarchyUniqueName](#): It allows to specify dimension unique name whose hierarchies alone should be used in the expression. This will be applicable only for calculated dimension.
- [formatString](#): It allows to set the format string for the resultant calculated field.

You need to set [isCalculatedField](#) property to true, while adding calculated fields to respective axis through code behind.

Also calculated fields can be added at run time through the built-in dialog. The dialog can be enabled by setting the [allowCalculatedField](#) property to **true** and by injecting the **CalculatedFieldService** module as follows. You will see a button enabled in the Field List UI automatically to invoke the calculated field dialog and perform necessary operation.

If the **CalculatedFieldService** module is not injected, the calculated field dialog will not be rendered with the pivot table component. Moreover calculated measure can be added only in value axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';

```

```

import { IDataOptions, IDataset, PivotView, FieldListService,
CalculatedFieldService } from '@syncfusion/ej2-angular-pivotview';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService, CalculatedFieldService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
allowCalculatedField='true' showFieldList='true'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
        this.dataSourceSettings = {
            catalog: 'Adventure Works DW 2008 SE',
            cube: 'Adventure Works',
            providerType: 'SSAS',
            enableSorting: true,
            url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
            localeIdentifier: 1033,
            rows: [
                { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
            ],
            columns: [
                { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
                { name: '[Measures]', caption: 'Measures' },
            ],
            values: [
                { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
                { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' },
                { name: 'Order on Discount', isCalculatedField: true }
            ],
            filters: [
                { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
            ],
            calculatedFieldSettings: [
                {
                    name: 'BikeAndComponents',
                    formula: '([Product].[Product
Categories].[Category].[Bikes] + [Product].[Product
Categories].[Category].[Components] )',
                    hierarchyUniqueName: '[Product].[Product Categories]',
                    formatString: 'Standard'
                },
                {
                    name: 'Order on Discount',

```

```

        formula: '[Measures].[Order Quantity] +
([Measures].[Order Quantity] * 0.10)',
        formatString: 'Currency'
    },
    ],
    filterSettings: [
        {
            name: '[Date].[Fiscal]', items:
[ '[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]' ],
            levelCount: 3
        }
    ]
};
this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Users can add a calculated field at runtime through the built-in dialog by using the following steps.

Step 1: Click the "CALCULATED FIELD" button in the field list dialog positioned at the top right corner. The calculated field dialog will be opened now. Enter the name of the calculated field to be created.

Field List**CALCULATED FIELD**

All Fields	Filters	Columns
<div> <div>Calculated Members</div> <div> <div>Measures</div> <div>Account</div> <div>Customer</div> <div>Date</div> <div>Delivery Date</div> <div>Department</div> <div>Destination Currency</div> </div> </div>	<div> <div>Date Fiscal (All P...</div> </div>	<div> <div>Product Categ...</div> <div>Measures</div> </div>
	<div> <div>Rows</div> <div>Customer Geo...</div> </div>	<div> <div>Values</div> <div>Customer Count</div> <div>Internet Sales Amount</div> </div>

CLOSE

CREATE CALCULATED FIELD

Field Name: BikeAndComponents

Expression: Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type: Measure

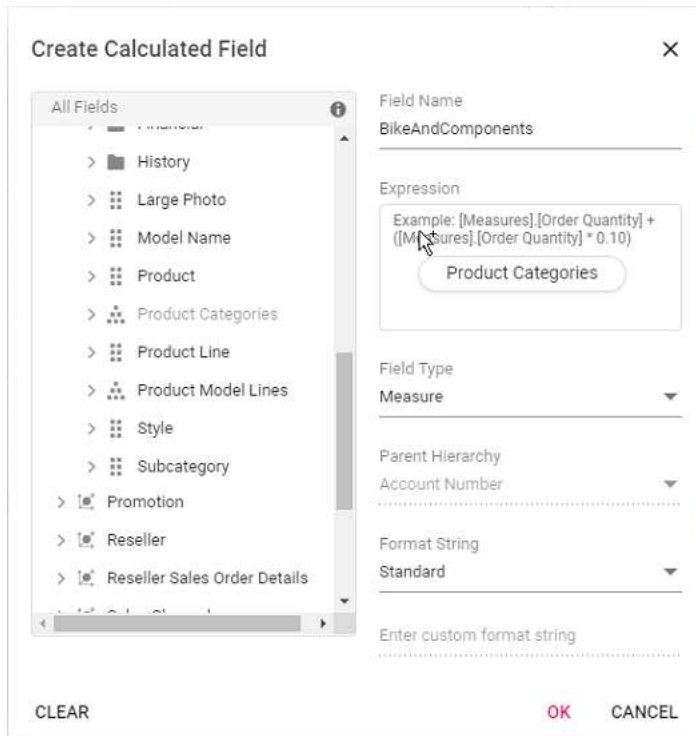
Parent Hierarchy: Account Number

Format String: Standard

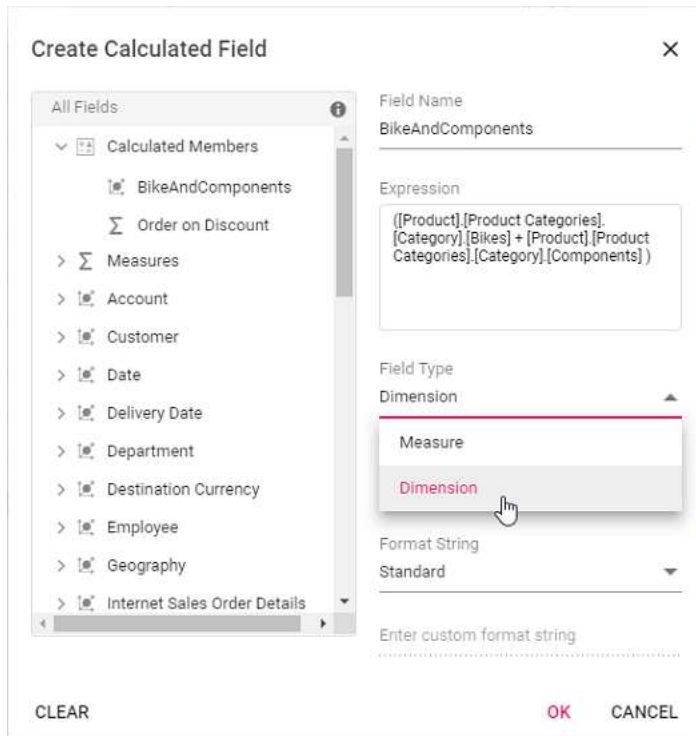
Enter custom format string

CLEAR OK CANCEL

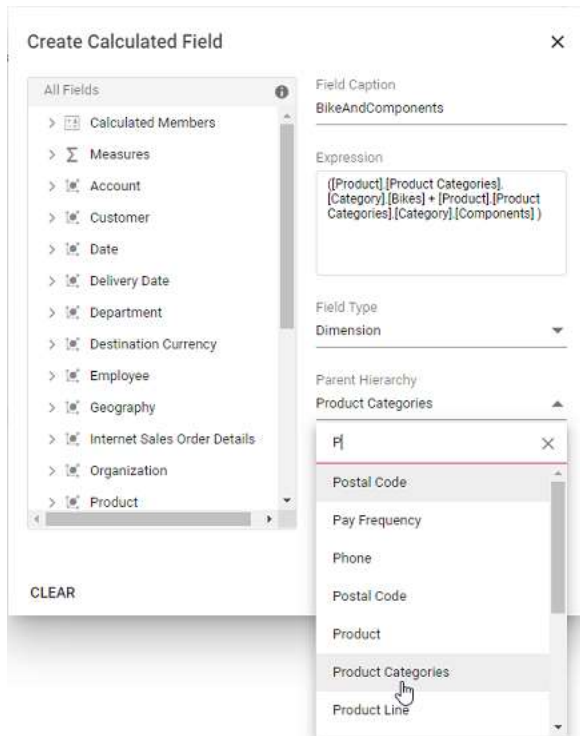
Step 2: Frame the expression by dragging and dropping the fields from the tree view on the left side of the dialog using simple arithmetic operators. **Example:** "IIF([Measures].[Internet Sales Amount]^0.5 > 100, [Measures].[Internet Sales Amount]*100, [Measures].[Internet Sales Amount]/100)". Please refer here to learn more about the supported [operators](#) and [functions](#) to frame the expression.



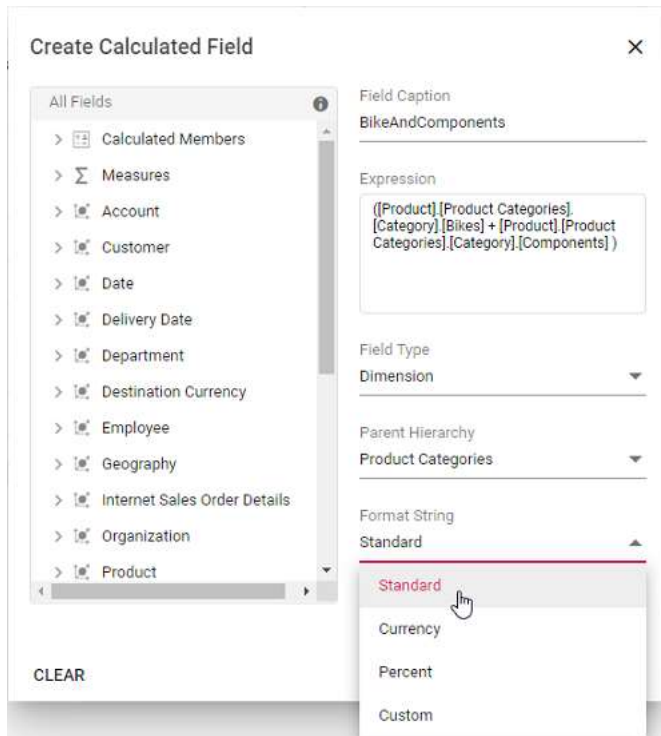
Step 3: Confirm the type of the field to be created - calculated measure or calculated dimension.



Step 4: Choose the parent hierarchy of the calculated field. NOTE: It is only applicable to the calculated dimension.



Step 5: Then select the format string from the drop-down list and finally click "OK".



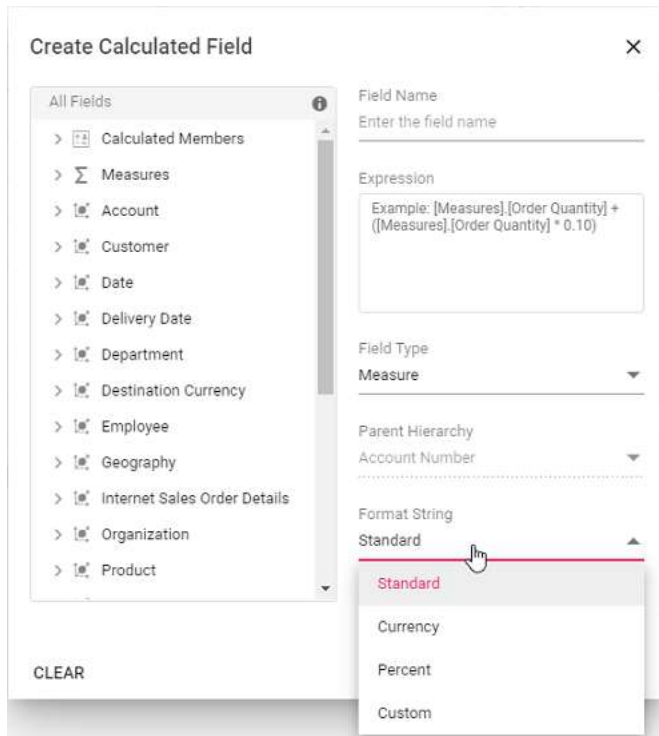
	Accessories		
	Customer Count	Internet Sales Amount	Order on Discount
► Australia	2,905	\$138,690.63	\$68,124.10
► Canada	1,230	\$103,377.85	\$68,124.10
► France	1,505	\$63,406.78	\$68,124.10
► Germany	1,535	\$62,232.59	\$68,124.10

Format String

Allows you to specify the required format string while creating new calculated field. Supported format strings are:

- **Standard** - Denotes the numeric type.
- **Currency** - Denotes the currency type.
- **Percent** - Denotes the percentage type.
- **Custom** - Denotes the custom format. For example: "###0.##0#". This shows the value "9584.3" as "9584.300."

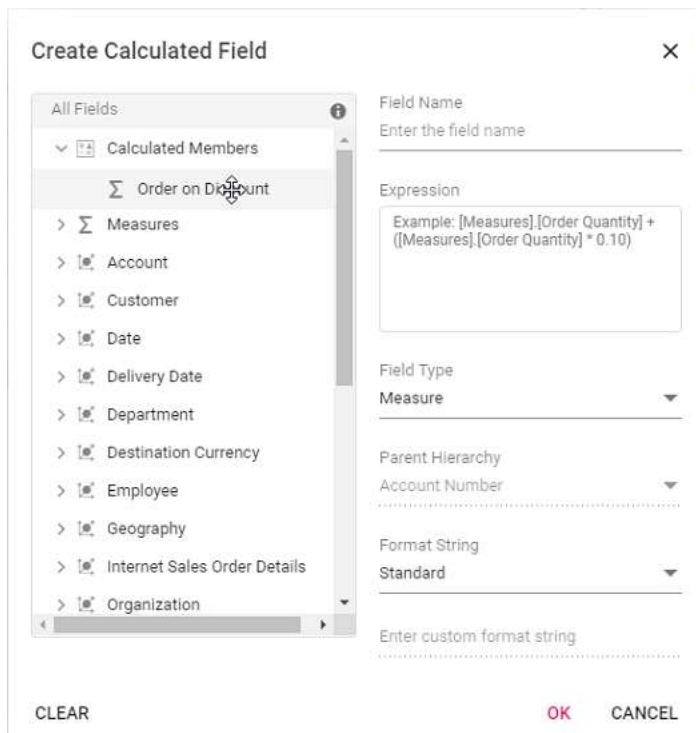
By default, **Standard** will be selected from the drop down list.



Renaming the existing calculated field

Existing calculated field can be renamed only through the UI at runtime. To do so, open the calculated field dialog, click the target field. User can now see the existing name getting displayed in the text box at the top of the dialog. Now, change the name based on user requirement and click "OK".

<!-- markdownlint-disable MD012 -->



Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption
Order Quantity on Discount

Expression
[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

[Editing the existing calculated field formula](#)

Existing calculated field formula can be edited only through the UI at runtime. To do so, open the calculated field dialog, click the target field. User can now see the existing expression getting displayed in a "Expression" section. Now, change the expression based on user requirement and click "OK".

Create Calculated Field

All Fields

Calculated Members

Σ Order on Discount

Σ Measures

Account

Customer

Date

Delivery Date

Department

Destination Currency

Employee

Geography

Internet Sales Order Details

Organization

Field Name

Enter the field name

Expression

Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type

Measure

Parent Hierarchy

Account Number

Format String

Standard

Enter custom format string

CLEAR

OK

CANCEL

Create Calculated Field

All Fields

Calculated Members

Σ Order on Discount

Σ Measures

Account

Customer

Date

Delivery Date

Department

Destination Currency

Employee

Geography

Internet Sales Order Details

Organization

Field Caption

Order Quantity on Discount

Expression

[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.50)

Field Type

Measure

Parent Hierarchy

Account Number

Format String

Standard

Enter custom format string

CLEAR

OK

CANCEL

Reusing the existing formula in a new calculate field

While creating a new calculated field, if user wants to add the formula of an existing calculated field, it can be done easily. To do so, simply drag-and-drop the existing calculated field to the "Expression" section.

Create Calculated Field

All Fields:

- Calculated Members
 - Order on Discount
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Name:
Enter the field name

Expression:
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type:
Measure

Parent Hierarchy:
Account Number

Format String:
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
 - Account
 - Customer
 - Date
 - Delivery Date
 - Department
 - Destination Currency
 - Employee
 - Geography
 - Internet Sales Order Details
 - Organization

Field Caption
Discount Quantity

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)
Order on Discount

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
 - Account
 - Customer
 - Date
 - Delivery Date
 - Department
 - Destination Currency
 - Employee
 - Geography
 - Internet Sales Order Details
 - Organization

Field Caption
Discount Quantity

Expression
[Measures].[Order on Discount]

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Modifying the existing format string

Existing calculated field's format string can be modified only through the UI at runtime. To do so, open the calculated field dialog and click the target calculated field. User can now see the format string for

the existing calculated field getting displayed in a drop-down list. Change the format string based on the requirement and finally click "OK".

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Name
Enter the field name

Expression
Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

Format String
Standard

Enter custom format string

CLEAR OK CANCEL

Create Calculated Field

All Fields

- Calculated Members
 - Order on Discount**
- Measures
- Account
- Customer
- Date
- Delivery Date
- Department
- Destination Currency
- Employee
- Geography
- Internet Sales Order Details
- Organization

Field Caption
Order on Discount

Expression
[Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

Field Type
Measure

Parent Hierarchy
Account Number

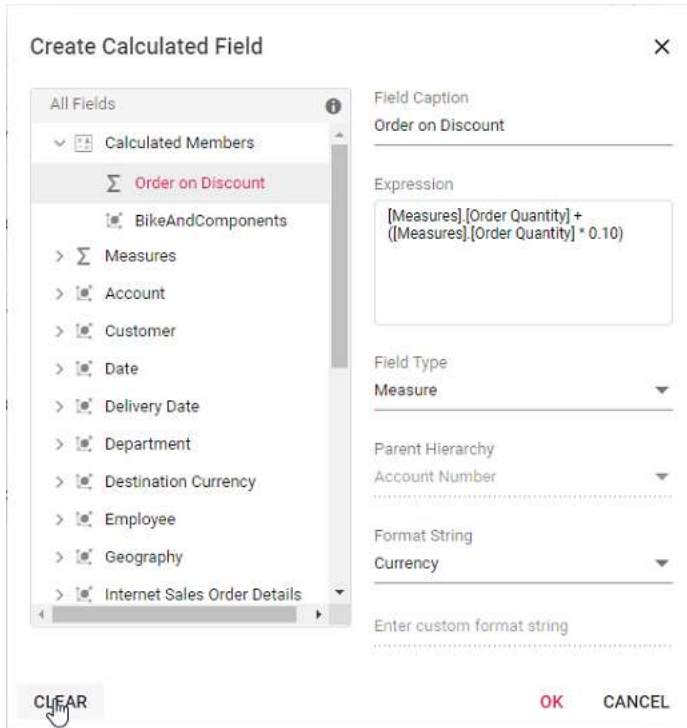
Format String
Currency

- Standard
- Currency**
- Percent
- Custom

CLEAR

Clearing the changes while editing the calculated field

Previous changes can be cleared by using the "Clear" option while performing operations such as creating and editing the calculated field. To do so, click the "Clear" button in the bottom left corner of the dialog.



Virtual Scrolling

Allows large amounts of data to be loaded without any performance degradation by rendering rows and columns in relation to the current viewport. Rest of the data will be brought into the viewport dynamically based on vertical or horizontal scroll position. This feature can be enabled by setting the [enableVirtualization](#) property to **true**.

To use the virtual scrolling feature, inject the **VirtualScroll** module into the pivot table.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService],
  // specifies the template string for the pivot table component
```

```

    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
enableVirtualization='true' [width]=width></ejs-pivotview>`
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
      this.dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
          { name: '[Customer].[Customer]', caption: 'Customer' },
        ],
        columns: [
          { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
          { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
          { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
          { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
          { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
          {
            name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
            levelCount: 3
          }
        ],
        formatSettings: [{ name: '[Measures].[Internet Sales Amount]',
format: 'C0' }]
      };
      this.width = "100%";
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Limitations for virtual scrolling

- In virtual scrolling, the [columnWidth](#) property in [gridSettings](#) should be in pixels, and percentage values are not accepted.
- Resizing columns or setting width to individual columns affects the calculation used to pick the correct page on scrolling.
- When using OLAP data, subtotals and grand totals are only displayed when measures are bound at the last position in the [rows](#) or [columns](#) axis. Otherwise, the data from the pivot table will be shown without summary totals.
- When the pivot table's width and height are large, the loading data count in the current, previous, and next viewports (pages) will also increase, affecting performance.

Run the application

The quickstart project is configured to compile and run the application in the browser. Use the following command to run the application.

```
`sh
ng serve --open
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
CalculatedFieldService } from '@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, CalculatedFieldService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
allowCalculatedField='true' showFieldList='true'></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
```

```

        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        calculatedFieldSettings: [
            {
                name: 'BikeAndComponents',
                formula: '([Product].[Product
Categories].[Category].[Bikes] + [Product].[Product
Categories].[Category].[Components] )',
                hierarchyUniqueName: '[Product].[Product Categories]',
                formatString: 'Standard'
            },
            {
                name: 'Order on Discount',
                formula: '[Measures].[Order Quantity] +
([Measures].[Order Quantity] * 0.10)',
                formatString: 'Currency'
            }
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Data Binding

To bind OLAP datasource to the pivot table, you need to specify following properties under [dataSourceSettings](#) option.

Properties	Description
cube	Points the respective cube name from OLAP database.
providerType	Points the provider type for pivot table to identify the type of data source.
url	Contains the cube URL for establishing the connection (online).
catalog	Contains the database name (catalog name) to fetch the data.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product
Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
    }
  }
}
```

```

        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Fields

Measures in row axis

By default, the measures are plotted in column axis. To plot those measures in row axis, place the **Measures** button in the row axis as follows.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

      PivotViewAllModule,
      PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component

```

```

    template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
      this.dataSourceSettings = {
        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
          { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
          { name: '[Measures]', caption: 'Measures' }
        ],
        columns: [
          { name: '[Product].[Product Categories]', caption: 'Product
Categories' }
        ],
        values: [
          { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
          { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
          { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
          {
            name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
            levelCount: 3
          }
        ]
      };
      this.width = "100%";
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Measures in different position

You can place measures in different position in row or column axis either thorough code behind or UI. In this sample, **measures** placed before the dimension in the column axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
 '@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
      ],
      columns: [
        { name: '[Measures]', caption: 'Measures' },
        { name: '[Product].[Product Categories]', caption: 'Product
Categories' }
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
        { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
      ],
      filters: [
        { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
      ],
      filterSettings: [
```



```

        {
            name: '[Date].[Fiscal]', items:
            ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
             '[Date].[Fiscal].[Fiscal Year].&[2005]'],
            levelCount: 3
        }
    ]
};
this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Named set

Named set is a multidimensional expression (MDX) that returns a set of dimension members, which can be created by combining the cube data, arithmetic operators, numbers, and functions.

You can bind the named sets in the pivot table by setting it's unique name in the [name](#) property either in row or column axis and [isNamedSet](#) boolean property to **true**. In this sample, we have added "Core Product Group" named set in the column axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {

```

```

        catalog: 'Adventure Works DW 2008 SE',
        cube: 'Adventure Works',
        providerType: 'SSAS',
        enableSorting: true,
        url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
        localeIdentifier: 1033,
        rows: [
            { name: '[Customer].[Customer Geography]', caption:
'Customer Geography' },
        ],
        columns: [
            {
                name: "[Core Product Group]",
                isNamedSet: true
            },
            { name: '[Measures]', caption: 'Measures' },
        ],
        values: [
            { name: '[Measures].[Customer Count]', caption: 'Customer
Count' },
            { name: '[Measures].[Internet Sales Amount]', caption:
'Internet Sales Amount' }
        ],
        filters: [
            { name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
        ],
        filterSettings: [
            {
                name: '[Date].[Fiscal]', items:
['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
                '[Date].[Fiscal].[Fiscal Year].&[2005]'],
                levelCount: 3
            }
        ]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Configuring authentication

Users can configure basic authentication information to access the OLAP cube using the [authentication](#) property. The settings required to configure are as follows:

- [userName](#): It allows the user to set a username that recognizes the basic authentication of the IIS.
- [password](#): It allows to set the appropriate password.

If the user does not configure the authentication, a default popup will appear in the browser to get the authentication information.

```
`typescript
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: <div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showFieldList='true'></ejs-
pivotview></div>
})
export class AppComponent {
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      providerType: 'SSAS',
      enableSorting: true,
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
        { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' }
      ],
    };
  }
}
```

```

filters: [
{ name: '[Date].[Fiscal]', caption: 'Date Fiscal' },
],
filterSettings: [
{
name: '[Date].[Fiscal]', items: ['[Date].[Fiscal].[Fiscal Quarter].&[2002]&[4]',
'[Date].[Fiscal].[Fiscal Year].&[2005]'],
levelCount: 3
}
],
authentication: {
userName: 'username',
password: 'password'
}
};
this.width = "100%";
}
}
,

```

Roles

SQL Server Analysis Services uses [roles](#) to limit data access within a cube. Each role defines a set of permissions that can be granted to a single user or groups of users. It is used to manage security by limiting access to sensitive data and determining who has access to and can change the cube. It can be configured using the [roles](#) property in [dataSourceSettings](#).

The [roles](#) property can be used to specify one or more roles to the OLAP cube, separated by commas.

```

`typescript
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-pivotview';
@Component({
selector: 'app-container',
providers: [],
// specifies the template string for the pivot table component
template: <div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width></ejs-pivotview></div>
})

```

```

export class AppComponent {
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      catalog: 'Adventure Works DW 2008 SE',
      cube: 'Adventure Works',
      roles: 'Role1',
      providerType: 'SSAS',
      url: 'https://bi.syncfusion.com/olap/msmdpump.dll',
      localeIdentifier: 1033,
      rows: [
        { name: '[Customer].[Customer Geography]', caption: 'Customer Geography' },
      ],
      columns: [
        { name: '[Product].[Product Categories]', caption: 'Product Categories' },
        { name: '[Measures]', caption: 'Measures' },
      ],
      values: [
        { name: '[Measures].[Customer Count]', caption: 'Customer Count' },
        { name: '[Measures].[Internet Sales Amount]', caption: 'Internet Sales Amount' }
      ]
    };
    this.width = "100%";
  }
}

```

OLAP Cube: Elements

Field list

The field list, aka cube dimension browser, is a tree view like structure that organizes the cube elements such as dimensions, hierarchies, measures, etc., from the selected cube into independent logical groups.

Types of node in field list

- **Display folder:** A folder that contains a set of similar elements.
- **Measure:** Quantity available for analysis.
- **Dimension:** A name given to the parts of the cube that categorizes data.

- **Attribute Hierarchy:** Level of attributes down the hierarchy.
- **User-defined Hierarchy:** Members of a dimension in a hierarchical structure.
- **Level:** Denotes a specific level in the category.
- **Named Set:** A collection of tuples and members, that can be defined and saved as a part of cube definition for later use.

Measure

In a cube, a measure is a set of values that are based on a column in the cube's fact table and are usually numeric. The measures are the central values of a cube that are analyzed. That is, measures are the numeric data of primary interest to users browsing a cube. You can select measures depend on the types of users request. Some common measures are sales, costs, expenditures, and production count.

Dimension

A simple dimension object is composed of basic information such as name, hierarchy, level, and members. You can create a dimension element by specifying its name and providing the hierarchy and level name. The dimension element contains the hierarchical details and information about each included level elements in that hierarchy. A hierarchy can have any number of level elements and the level elements can have any number of members and the member elements can have any number of child members.

Hierarchy

Each element of a dimension can be summarized using a hierarchy. The hierarchy is a series of parent-child relationship, where a parent member represents the consolidation of members which are its children. Parent members can be further aggregated as the children of another parent. For example, May 2005 can be summarized into Second Quarter 2005 which in turn would be summarized in the year 2005.

Level

Level element is the child of hierarchy element which contains a set of members, each of which has the same rank within a hierarchy.

Attribute hierarchy

Attribute hierarchy contains the following levels:

- A leaf level contains distinct attribute member, and each member of the leaf level is known as a leaf member.
- Intermediate levels if the attribute hierarchy is a parent-child hierarchy.
- An optional (all) level contains the aggregated value of the attribute hierarchy's leaf members, with the member of the (all) level also known as the (all) member.

User-defined hierarchy

User-defined hierarchy organizes the members of a dimension into hierarchical structure and provides navigation paths in a cube. For example, take a dimension table that supports three attributes such as year, quarter, and month. The year, quarter, and month attributes are used to construct a user-defined hierarchy, named Calendar, in the time dimension that relates to all levels.

Differentiating user-defined hierarchy and attribute hierarchy

- User-defined hierarchy contains more than one level whereas attribute hierarchy contains only one level.

- User-defined hierarchy provides the navigation path between the levels taken from attribute hierarchies of the same dimension.
- The attribute hierarchy and the user-defined hierarchy are represented in different ways as shown in the following table.

Named set

A named set is a collection of tuples and members, which can be defined and saved as a part of the cube definition. Named set records reside inside the sets folder, which is under a dimension element. These elements can be dragged to [rows](#) or [columns](#) axis via grouping bar or field list at runtime. To work with a lengthy, complex, or commonly used expression easier, Multidimensional Expressions (MDX) allows you to define a named set.










Calculated field

The calculated field allows user to insert or add a new calculated field based on the available OLAP cube elements from the bound data source. Calculated fields are nothing but customized dimensions or measures that are newly created based on the user-defined expression.

The two types of calculated fields are as follows:

- **Calculated Measure** – Creates a new measure through user-defined expression.
- **Calculated Dimension** – Creates a new dimension through user-defined expression.

Symbolic representation of the nodes inside field list

Icon	Name	Node type	Is Draggable
-----	-----	-----	-----
	Display Folder	Display Folder	False
	Measure	Measure	False
	Dimension	Dimension	False
	User Defined Hierarchy	Hierarchy	True
	Attribute Hierarchy	Hierarchy	True
  	Levels (in order)	Level Element	True
	Named Set	Named Set	True

In general, the Pivot Table is created using the built-in engine for given data source. This is an optional feature that allows you to create the Pivot Table with a server-side pivot engine and external data binding. And this option is applicable only for relational data source.

Server side pivot engine in Angular Pivotview component

This section briefs the Syncfusion assembly [Syncfusion.Pivot.Engine](#), which is used in a server-side application to perform all Pivot calculations such as aggregation, filtering, sorting, grouping, and so on,

and only the information to be displayed in the Pivot Table's viewport is passed to the client-side (browser) via web service (Web API) rather than the entire data source. It reduces network traffic and improves the rendering performance of the Pivot Table, especially when dealing with large amounts of data. It also works best with virtual scrolling enabled and supports all the Pivot Table's existing features.

Quick steps to render the Pivot Table by using the server-side Pivot Engine

[Download and installing Server-side Pivot Engine](#)

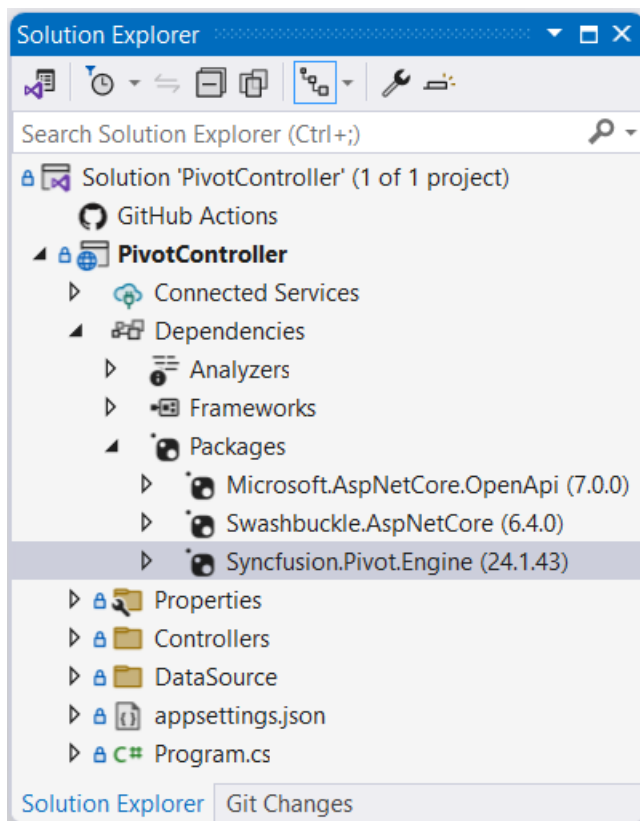
1. Download the ASP.NET Core-based stand-alone Pivot Table [application](#) from the GitHub repository.

2. The **PivotController** (Server-side) application that is downloaded includes the following files.

- **PivotController.cs** file under **Controllers** folder – This helps to do data communication with Pivot Table.
- **DataSource.cs** file under **DataSource** folder – This file has model classes to define the structure of the data sources.
- The sample data source files **sales.csv** and **sales-analysis.json** under **DataSource** folder.

3. Open the **PivotController** application in Visual Studio where the Syncfusion library

[Syncfusion.Pivot.Engine](#) will be downloaded automatically from the nuget.org site.



[Connecting Pivot Table to Server-side Pivot Engine](#)

1. Run the **PivotController** (Server-side) application which will be hosted in IIS shortly.

2. Then in the Pivot Table sample, set the [mode](#) property under [dataSourceSettings](#) as **Server** and map the URL of the hosted Server-side application in [URL](#) property of [dataSourceSettings](#).

`javascript


```

import { Component, OnInit } from '@angular/core';
import { IDataOptions } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-container',
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings ></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      url: 'https://localhost:44350/api/pivot/post',
      mode: 'Server',
    };
  }
}

```

3. Frame and set the report based on the data source available in the **PivotController** application.

```

`javascript
import { Component, OnInit } from '@angular/core';
import { IDataOptions } from '@syncfusion/ej2-angular-pivotview';
@Component({
  selector: 'app-container',
  template: <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings ></ejs-pivotview>
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      url: 'https://localhost:44350/api/pivot/post',
      mode: 'Server',
    };
  }
}

```

```

rows: [{
  name: 'ProductID', caption: 'Product ID'
}],
formatSettings: [{
  name: 'Price', format: 'C'
}],
columns: [{
  name: 'Year', caption: 'Production Year'
}],
values: [
  { name: 'Sold', caption: 'Units Sold' },
  { name: 'Price', caption: 'Sold Amount' }
],
];
}
}
,

```

4. Run the sample to get the following result.

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7375140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7392240
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7880150

Available configurations in Server-side application

[Supportive Data Sources](#)

The server-side Pivot Engine supports the following data sources,

- Collection
- JSON
- CSV
- DataTable
- Dynamic

Collection

The collection data sources such as List, IEnumerable, and so on are supported. This can be bound using the **GetData** controller method. Also, in the Pivot Table sample, set the [type](#) property under [dataSourceSettings](#) to **JSON**, which is also the default enumeration value.

In the server-side application (**PivotController**), a collection type data source is framed in the **DataSource.cs** file as shown in the following.

```
`csharp
public class PivotViewData
{
    public string ProductID { get; set; }
    public string Country { get; set; }
    public string Product { get; set; }
    public double Sold { get; set; }
    public double Price { get; set; }
    public string Year { get; set; }
    public List<PivotViewData> GetVirtualData()
    {
        List<PivotViewData> VirtualData = new List<PivotViewData>();
        for (int i = 1; i <= 10000; i++)
        {
            PivotViewData p = new PivotViewData
            {
                ProductID = "PRO-" + ((100 + i)%20),
                Year = (new string[] { "FY 2015", "FY 2016", "FY 2017", "FY 2018", "FY 2019" })[new Random().Next(5)],
                Country = (new string[] { "Canada", "France", "Australia", "Germany", "France" })[new
                Random().Next(5)],
                Product = (new string[] { "Car", "Van", "Bike", "Flight", "Bus" })[new Random().Next(5)],
                Price = (3.4 * i) + 500,
                Sold = (i * 15) + 10
            };
            VirtualData.Add(p);
        }
        return VirtualData;
    }
}
```

`

To bind the data source, set its model type **PivotViewData** to **TValue** of the **PivotEngine** class.

`csharp

```
private PivotEngine<DataSource.PivotViewData> PivotEngine = new  
PivotEngine<DataSource.PivotViewData>();
```

`

Then call the data source in **GetData** method of **PivotController.cs** file.

`csharp

```
public async Task<object> GetData(FetchData param)  
{  
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,  
        async (cacheEntry) =>  
        {  
            cacheEntry.SetSize(1);  
            cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);  
            // Here bind the collection type data source.  
            return new DataSource.PivotViewData().GetVirtualData();  
        });  
}
```

`

Finally set the appropriate report to the Pivot Table sample based on the above data source.

`javascript

```
this.dataSourceSettings = {  
    url: 'https://localhost:44350/api/pivot/post',  
    mode: 'Server',  
    rows: [{  
        name: 'ProductID', caption: 'Product ID'  
    }],  
    formatSettings: [{  
        name: 'Price', format: 'C'  
    }],  
    columns: [{  
        name: 'Year', caption: 'Production Year'    }]
```

```

}},
values: [
{ name: 'Sold', caption: 'Units Sold' },
{ name: 'Price', caption: 'Sold Amount' }
],
};

```

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7375140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7396240
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7880150

JSON

The JSON data from a local *.json file type can be connected to the Pivot Table. Here, the file can be read by the **StreamReader** option, which will give the result in the string format. The resultant string needs to be converted to collect data that can be bound to the Server-side pivot engine.

In the Server-side application, **sales-analysis.json** file is available under **DataSource** folder and its model type is defined in **DataSource.cs** file.

```

`csharp
public class PivotJSONData
{
    public string Date { get; set; }
    public string Sector { get; set; }
    public string EnerType { get; set; }
    public string EneSource { get; set; }
    public int PowUnits { get; set; }
    public int ProCost { get; set; }
    public List<PivotJSONData> ReadJSONData(string url)
    {
        WebClient myWebClient = new WebClient();
        Stream myStream = myWebClient.OpenRead(url);
        StreamReader stream = new StreamReader(myStream);
    }
}

```

```

string result = stream.ReadToEnd();
stream.Close();
return Newtonsoft.Json.JsonConvert.DeserializeObject<List<PivotJSONData>>(result);
}
}
`

```

To bind the data source, set its model type **PivotJSONData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotJSONData> PivotEngine = new PivotEngine<DataSource.
PivotJSONData>();
`

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
async (cacheEntry) =>
{
cacheEntry.SetSize(1);
cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
// Here bind JSON type data source from the sales-analysis.json file.
return new DataSource.PivotJSONData().ReadJSONData(_hostingEnvironment.ContentRootPath +
"//DataSource//sales-analysis.json");
});
}
`

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```

`javascript
this.dataSourceSettings = {
url: 'https://localhost:44350/api/pivot/post',
mode: 'Server',
type: 'JSON',
rows: [{
name: 'EneSource', caption: 'Energy Source'

```

```

}},
formatSettings: [{
  name: 'ProCost', format: 'C'
}],
columns: [{
  name: 'EnerType', caption: 'Energy Type'
}],
values: [
  { name: 'PowUnits', caption: 'Units Sold' },
  { name: 'ProCost', caption: 'Sold Amount' }
],
}
,

```

	Biomass		Free Energy		Grand Total
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
Bio-diesel	1042	\$1,439.00			1042
Ethanol Fuel	595	\$1,031.00			595
Geo-thermal			1528	\$2,115.00	1528
Hydro-electric			3378	\$3,244.00	3378
Solar			7929	\$6,210.00	7929
Wastage	712	\$1,043.00			712
Wind			15571	\$7,666.00	15571

JSON data from any remote server, like a local JSON file, can also be supported. It accepts both directly downloadable files (.json) and web service URLs. To bind this, the URL of the .json file of a remote server has to be mapped under the **GetData** method. The rest of the configurations are the same as described above.

In the server-side application, the CDN link is used to connect the same **sales-analysis.json** file which is already hosted in the Syncfusion server.

```
`csharp
```

```

public async Task<object> GetData(FetchData param)
{
  return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
      cacheEntry.SetSize(1);
      cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
    }
  );
}

```

```
// Here bind JSON type data source from remote server.

return new DataSource.PivotJSONData().ReadJSONData("http://cdn.syncfusion.com/data/sales-
analysis.json");
});
}
`
```

CSV

The CSV data from a local *.csv file type can be connected to the Pivot Table. Here, the file can be read by the **StreamReader** option, which will give the result in the string format. The resultant string needs to be converted to collect data that can be bound to the server-side pivot engine. Also, in the Pivot Table sample, set the [type](#) property under [dataSourceSettings](#) as **CSV**.

In the server application, the **sales.csv** file is available under the **DataSource** folder, and its model type is defined in the **DataSource.cs** file.

```
`csharp

public class PivotCSVData
{
    public string Region { get; set; }
    public string Country { get; set; }
    public string ItemType { get; set; }
    public string SalesChannel { get; set; }
    public string OrderPriority { get; set; }
    public string OrderDate { get; set; }
    public int OrderID { get; set; }
    public string ShipDate { get; set; }
    public int UnitsSold { get; set; }
    public double UnitPrice { get; set; }
    public double UnitCost { get; set; }
    public double TotalRevenue { get; set; }
    public double TotalCost { get; set; }
    public double TotalProfit { get; set; }
    public List<string[]> ReadCSVData(string url)
    {
        List<string[]> data = new List<string[]>();
        using (StreamReader reader = new StreamReader(new WebClient().OpenRead(url)))
        {
```



```

string line;
while ((line = reader.ReadLine()) != null)
{
    line = line.Trim();
    if (!string.IsNullOrEmpty(line))
    {
        data.Add(line.Split(','));
    }
}
return data;
}
}

```

To bind the data source, set its model type **PivotCSVData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotCSVData> PivotEngine = new PivotEngine<DataSource.
PivotCSVData>();

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind CSV type data source from sales.csv file.
        return new DataSource.PivotCSVData().ReadCSVData(_hostingEnvironment.ContentRootPath +
        "//DataSource//sales.csv");
    });
}

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
this.dataSourceSettings = {
url: 'https://localhost:44350/api/pivot/post',
mode: 'Server',
type: 'CSV',
rows: [{
name: 'ItemType', caption: 'Item Type'
}],
formatSettings: [{
name: 'UnitPrice', format: 'C'
}],
columns: [{
name: 'Region'
}],
values: [
{ name: 'UnitsSold', caption: 'Units Sold' },
{ name: 'UnitPrice', caption: 'Sold Amount' }
],
}
```

	Asia		Australia and Oceania		Central Ar
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
Baby Food	55810	\$3,318.64	21548	\$765.84	
Beverages	84400	\$806.65	34486	\$332.15	
Cereal	32668	\$1,439.90	43195	\$1,645.60	
Clothes	24290	\$764.96	17020	\$655.68	
Cosmetics	84630	\$7,432.40	35220	\$3,060.40	
Fruits	35679	\$74.64	29951	\$46.65	
Household	28166	\$4,009.62	55444	\$6,014.43	

CSV data from any remote server, like a local CSV file, can also be supported. It accepts both directly downloadable files (.csv) and web service URLs. To bind this, the URL of the .csv file of a remote server has to be mapped under **GetData** method. The rest of the configurations are the same as described above.

In the server application, the CDN link is used to connect the same **sales.csv** file which is already hosted in the Syncfusion server.

```
`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind CSV type data source from remote server.
        return new DataSource.PivotCSVData().ReadCSVData("http://cdn.syncfusion.com/data/sales-
        analysis.csv");
    });
}
`
```

DataTable

In the server-side application, there is a manually created DataTable **BusinessObjectsDataView** by mapping the model type **PivotViewData** in **DataSource.cs** file.

```
`csharp
public class BusinessObjectsDataView
{
    public DataTable GetDataTable()
    {
        DataTable dt = new DataTable("BusinessObjectsDataTable");
        PropertyDescriptorCollection pdc = TypeDescriptor.GetProperties(typeof(PivotViewData));
        foreach (PropertyDescriptor pd in pdc)
        {
            dt.Columns.Add(new DataColumn(pd.Name, pd.PropertyType));
        }
        List<PivotViewData> list = new PivotViewData().GetVirtualData();
        foreach (PivotViewData bo in list)
        {
            DataRow dr = dt.NewRow();
```

```
foreach (PropertyDescriptor pd in pdc)
{
    dr[pd.Name] = pd.GetValue(bo);
}
dt.Rows.Add(dr);
}
return dt;
}
}
```

To bind the data source, set its model type **PivotViewData** to **TValue** of the **PivotEngine** class.

```
`csharp
private PivotEngine<DataSource.PivotViewData> PivotEngine = new
PivotEngine<DataSource.PivotViewData>();
,
```

Then call the data source in **GetData** method of **PivotController.cs** file.

```
`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind the DataTable.
        return new DataSource.BusinessObjectsDataView().GetDataTable();
    });
}
,
```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
this.dataSourceSettings = {
    url: 'https://localhost:44350/api/pivot/post',
```

```

mode: 'Server',
rows: [{
name: 'ProductID', caption: 'Product ID'
}],
formatSettings: [{
name: 'Price', format: 'C'
}],
columns: [{
name: 'Year', caption: 'Production Year'
}],
values: [
{ name: 'Sold', caption: 'Units Sold' },
{ name: 'Price', caption: 'Sold Amount' }
],
}
,

```

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
PRO-10	7901220	\$1,841,712.00	7561880	\$1,762,804.00	7591220
PRO-11	7174625	\$1,673,533.00	7338175	\$1,711,599.80	7164625
PRO-12	9168140	\$2,135,848.80	7689400	\$1,792,704.00	7375140
PRO-13	6674280	\$1,560,619.20	8585260	\$2,001,738.40	6574280
PRO-14	7489240	\$1,745,841.20	7167840	\$1,675,479.20	7392400
PRO-15	8149185	\$1,902,397.00	7450585	\$1,734,093.00	7249185
PRO-16	7070150	\$1,652,838.40	8037500	\$1,873,597.60	7880150

Dynamic

The model type has to be defined in the aforementioned data sources. However, there is no need to define a model type for the following data sources, which are also supported by the server-side pivot engine.

ExpandoObject

In the server-side application, an **ExpandoObject** type data source is available under the class **PivotExpandoData** in **DataSource.cs** file.

```

`csharp
public class PivotExpandoData
{
public List<ExpandoObject> Orders { get; set; } = new List<ExpandoObject>();

```

```

public List<ExpandoObject> GetExpandoData()
{
    Orders = Enumerable.Range(1, 75).Select((x) =>
    {
        dynamic d = new ExpandoObject();
        d.OrderID = 1000 + (x % 100);
        d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
        Random().Next(5)];
        d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
        d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new DateTime(2018, 10, 3), new
        DateTime(1995, 9, 9), new DateTime(2012, 8, 2), new DateTime(2015, 4, 11) })[new Random().Next(5)];
        d.ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)];
        d.Verified = (new bool[] { true, false })[new Random().Next(2)];
        return d;
    }).Cast<ExpandoObject>().ToList<ExpandoObject>();
    return Orders;
}

```

To bind the data source, set its model type as **ExpandoObject** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<ExpandoObject> PivotEngine = new PivotEngine<ExpandoObject>();

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here returns ExpandoObject type data source.
        return new DataSource.PivotExpandoData().GetExpandoData();
    }
}

```

```
});
}
,
```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```
`javascript
this.dataSourceSettings = {
url: 'https://localhost:44350/api/pivot/post',
mode: 'Server',
rows: [{
name: 'CustomerID', caption: 'Customer ID'
}],
columns: [{
name: 'ShipCountry', caption: 'Ship Country'
}],
values: [
{ name: 'Freight', caption: 'Units Sold' }
],
}
,
```

	UK	USA	Grand Total
	Units Sold	Units Sold	Units Sold
ALFKI	1187	738	1925
ANANTR	1011	814	1825
ANTON	1046	632	1678
BLONP	1080	749	1829
BOLID	492	929	1421
Grand Total	4816	3862	8678

Dynamic Objects

In the server-side application, a data source is framed by dynamic objects which is available under the class **PivotDynamicData** in the **DataSource.cs** file.

```
`csharp
public class PivotDynamicData
{
public List<DynamicDictionary> Orders = new List<DynamicDictionary>() { };
public List<DynamicDictionary> GetDynamicData()
```

```

{
Orders = Enumerable.Range(1, 100).Select((x) =>
{
dynamic d = new DynamicDictionary();
d.OrderID = 100 + x;
d.CustomerID = (new string[] { "ALFKI", "ANANTR", "ANTON", "BLONP", "BOLID" })[new
Random().Next(5)];
d.Freight = (new double[] { 2, 1, 4, 5, 3 })[new Random().Next(5)] * x;
d.OrderDate = (new DateTime[] { new DateTime(2010, 11, 5), new DateTime(2018, 10, 3), new
DateTime(1995, 9, 9), new DateTime(2012, 8, 2), new DateTime(2015, 4, 11) })[new Random().Next(5)];
d.ShipCountry = (new string[] { "USA", "UK" })[new Random().Next(2)];
d.Verified = (new bool[] { true, false })[new Random().Next(2)];
return d;
}).Cast<DynamicDictionary>().ToList<DynamicDictionary>();
return Orders;
}

public class DynamicDictionary : System.Dynamic.DynamicObject
{
Dictionary<string, object> dictionary = new Dictionary<string, object>();
public override bool TryGetMember(GetMemberBinder binder, out object result)
{
string name = binder.Name;
return dictionary.TryGetValue(name, out result);
}
public override bool TrySetMember(SetMemberBinder binder, object value)
{
dictionary[binder.Name] = value;
return true;
}
}

//The "GetDynamicMemberNames" method of the "DynamicDictionary" class must be overridden and
return the property names to perform data operation and editing while using dynamic objects.
public override System.Collections.Generic.IEnumerable<string> GetDynamicMemberNames()
{
return this.dictionary?.Keys;
}

```



```

}
}
}
`

```

To bind the data source, set its class **PivotDynamicData** to **TValue** of the **PivotEngine** class.

```

`csharp
private PivotEngine<DataSource.PivotDynamicData> PivotEngine = new
PivotEngine<DataSource.PivotDynamicData>();
`

```

Then call the data source in **GetData** method of **PivotController.cs** file.

```

`csharp
public async Task<object> GetData(FetchData param)
{
    return await _cache.GetOrCreateAsync("dataSource" + param.Hash,
    async (cacheEntry) =>
    {
        cacheEntry.SetSize(1);
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        // Here bind data source with dynamic objects.
        return new DataSource.PivotDynamicData().GetDynamicData();
    });
}
`

```

Finally set the appropriate report to the Pivot Table sample based on the above data source.

```

`javascript
this.dataSourceSettings = {
    url: 'https://localhost:44350/api/pivot/post',
    mode: 'Server',
    rows: [{
        name: 'CustomerID', caption: 'Customer ID'
    }],
    columns: [{
        name: 'ShipCountry', caption: 'Ship Country'
    }]
}

```

```

}},
values: [
{ name: 'Freight', caption: 'Units Sold' }
],
}
,

```

	UK	USA	Grand Total
	Units Sold	Units Sold	Units Sold
ALFKI	1030	742	1772
ANANTR	1352	1018	2370
ANTON	2525	1782	4307
BLONP	1786	1002	2788
BOLID	1409	2012	3421
Grand Total	8102	6556	14658

Controller Configuration

Memory Cache

In the server-side application, the [Memory Cache](#) option is used to store the data source and engine properties in RAM, which will be used for UI operations. To improve performance, this limits the execution of all initial rendering code to regenerate the aggregated values during each UI operation. The codes below show how we use the memory cache option in the **GetEngine** method to store engine properties.

```

`csharp
public async Task<EngineProperties> GetEngine(FetchData param)
{
    isRendered = false;
    // Engine properties are stored in memory cache with GUID "param.Hash".
    return await _cache.GetOrCreateAsync("engine" + param.Hash,
    async (cacheEntry) =>
    {
        isRendered = true;
        cacheEntry.SetSize(1);
        // Memory cache expiration time can be set here.
        cacheEntry.AbsoluteExpiration = DateTimeOffset.UtcNow.AddMinutes(60);
        PivotEngine.Data = await GetData(param);
        return await PivotEngine.GetEngine(param);
    });
}

```

```
}
,
```

The engine properties are stored in RAM as a cache with a unique ID (GUID) that is transferred from the client-side source code. The GUID is generated at random and will be changed if the page containing the Pivot Table is refreshed or opened in a new tab/window. As a result, each GUID's memory cache contains unique information, and the component operates independently.

Meanwhile, the memory cache is set to expire after 60 minutes from RAM to free its memory. If the component is still running, the data will be generated and stored for another 60 minutes.

Methods and its needs

- **Post:** Allows to get the information from the client-side source and calls appropriate controller methods.
- **GetEngine:** Allows to store the engine properties in RAM as a cache which fires on initial rendering or when the memory cache is expired.
- **GetData:** Allows to store data source in RAM as a cache which fires on initial rendering or when the memory cache is expired.
- **GetMembers:** Allows to get the members of a field. This fires when the member editor is opened to do a filtering operation.
- **GetRawData:** Allows to get raw data of an aggregated value cell. This fires when the drill-through or editing dialog is opened.
- **GetPivotValues:** Allows to update the stored engine properties in-memory cache and returns the aggregated values to browser to render the Pivot Table. Here, the return value can be modified. The Pivot Table will be rendered browser-based on this.

Pivot chart in Angular Pivotview component

In pivot table component, pivot chart would act as an additional visualization component with its basic and important characteristic like drill down and drill up, 15+ chart types, series customization, axis customization, legend customization, export, print and tooltip. Its main purpose is to show the pivot data in graphical format.

If user prefers, the pivot chart component can also be displayed individually with pivot values and can change the report dynamically with the help of field list and grouping bar. Using the [displayOption](#) property in pivot table, user can set the visibility of grid and chart in pivot table component. It holds below properties,

- [view](#): Specifies the pivot table component to display grid alone or chart alone or both.
- [primary](#): Specifies the pivot table to display either grid or chart as primary component during initial loading. It is applicable only when setting the property [view](#) to **Both**.

You need to inject the `PivotChartService` module to make pivot chart features available in the pivot table.

The below sample displays the pivot chart component based on the pivot report bound on it.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Column' } } as
ChartSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Data Binding

End user can bind both local and remote data binding options available in the component to feed the data. The [dataSource](#) property can be assigned either with an instance of [DataManager](#) or JavaScript object array collection.

For more information [refer](#) here.

Chart Types

Supports 21 different types of charts as follows,

- Line
- Column
- Area
- Bar
- StepArea
- StackingLine
- StackingColumn
- StackingArea
- StackingBar
- StepLine
- Pareto
- Bubble
- Scatter
- Spline
- SplineArea
- StackingLine100
- StackingColumn100
- StackingBar100
- StackingArea100
- Polar
- Radar

Line is the default pivot chart type. User can change the pivot chart type by using the property [type](#) in [chartSeries](#).

In the below code sample, the pivot chart type is set as **Bar**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [PivotChartService],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = { chartSeries: { type: 'Bar' } } as
ChartSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Accumulation Charts

Supports 4 different types of accumulation charts as follows,

- Pie
- Doughnut
- Funnel
- Pyramid

As like other chart types it can be changed using the property [type](#) in [chartSeries](#).

In the below code sample, the **Pie** chart is rendered, and the other accumulation charts can be switched using the drop-down list.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { DropDownList, ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
import { Pivot_Data } from './datasource';
@Component({
imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
template: `<div id="dropdown-control" style="margin-bottom:5px;">
    <table style="width: 350px;margin-left: 50px;">
        <tbody>
            <tr style="height: 50px">
                <td>
                    <div><b>Accumulation Chart:</b>
                    </div>
                </td>
                <td>
                    <div>
                        <select id="charttypes" name="ddl-view-
mode">
                            <option value='Pie'>Pie</option>
                            <option
value='Doughnut'>Doughnut</option>
                            <option value='Funnel'>Funnel</option>
                            <option value='Pyramid'>Pyramid</option>
                        </select>
                    </div>
                </td>
            </tr>
        </tbody>
    </table>
    </div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    public chartTypesDropDown?: DropDownList;
    @ViewChild('pivotview', { static: false })

```

```

public pivotGridObj?: PivotViewComponent;
onChange(args: ChangeEventArgs): void {
    this.chartSettings = { chartSeries: { type: args.value } } as
ChartSettings;
}
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year' }, { name: 'Products' }],
        rows: [{ name: 'Country' }, { name: 'Quarter' }],
        formatSettings: [{ name: 'Amount', format: 'C' }],
        values: [{ name: 'Amount' }, { name: 'Sold' }]
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Pie' } } as
ChartSettings;
    this.chartTypesDropDown = new DropDownList({
        floatLabelType: 'Auto',
        change: this.onChange.bind(this)
    });
    this.chartTypesDropDown.appendTo('#charttypes');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Drill Down/Up

In the accumulation charts, drill down and drill up operations can be performed using the built-in context menu option. It will be shown while clicking on the chart series. The context menu has the following options:

- **Expand** - It is to drill down the corresponding series until the last level.
- **Collapse** - It is to drill up the corresponding series until the first level.
- **Exit** - It is to close the context menu.

The drill operation in accumulation charts can be performed only for row headers.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';

```



```

import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public pivotData?: IDataset[];
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotViewComponent;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Products' }],
      rows: [{ name: 'Country' }, { name: 'Year' }, { name: 'Quarter'
    }],
      formatSettings: [{ name: 'Amount', format: 'C' }],
      values: [{ name: 'Amount' }, { name: 'Sold' }]
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Pie' } } as
ChartSettings;
  }
}

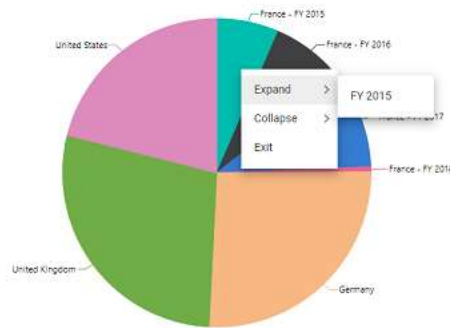
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Column Headers and Delimiters

Unlike other chart types, the accumulation charts consider the values of a single column from the pivot table to be drawn. Preferably the first column of the pivot table is considered by default. But it can be changed by defining the column headers using the `columnHeader` property in [chartSettings](#).

If the column has more than one header, then need to mention all the headers separated by the delimiter -, for example, **Germany-Road Bikes**. Using the property `columnDelimiter` in [chartSettings](#), one can set the desired delimiter to separate the column headers.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
```

```

ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['Germany'] }],
        columns: [{ name: 'Country' }, { name: 'Products' }],
        rows: [{ name: 'Year' }, { name: 'Quarter' }],
        formatSettings: [{ name: 'Amount', format: 'C' }],
        values: [{ name: 'Amount' }, { name: 'Sold' }]
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
        columnHeader: 'Germany-Road Bikes',
        columnDelimiter: '-',
        chartSeries: { type: 'Doughnut' }
    } as ChartSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label Customization

The data labels are visible by default showing header name. Its visibility can be modified using the **visible** boolean property in **dataLabel**. With regard to the label arrangement, the **Smart Labels** options help to arrange labels efficiently without overlapping. It can be disabled by setting the **enableSmartLabels** property in **chartSettings** as **false**.

The **position** property in **dataLabel** allows to specify the position of the data label. The available options are,

- **Outside**: Positions the label outside the point. It is the default option.
- **Inside**: Positions the label inside the point.

In the following code sample, the data labels are placed inside.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [PivotChartService],
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
  })
  export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year' }, { name: 'Products' }],
        rows: [{ name: 'Country' }, { name: 'Quarter' }],
        formatSettings: [{ name: 'Amount', format: 'C' }],
        values: [{ name: 'Amount' }, { name: 'Sold' }]
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = {
        enableSmartLabels: false,
        chartSeries: { dataLabel: { visible: true, position: 'Inside' },
type: 'Pyramid' }
      } as ChartSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The **Connector Line** will be visible when the data label is placed outside the chart. It can be customized using the `connectorStyle` property in `dataLabel` for its color, length, width etc. In the following code sample, the connector line is customized.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';

```

```

import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year' }, { name: 'Products' }],
            rows: [{ name: 'Country' }, { name: 'Quarter' }],
            formatSettings: [{ name: 'Amount', format: 'C' }],
            values: [{ name: 'Amount' }, { name: 'Sold' }]
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = {
            chartSeries: {
                dataLabel: { visible: true, position: 'Outside',
connectorStyle: { length: '50px', width: 2, dashArray: '5,3', color:
'#f4429e' } },
                type: 'Funnel'
            }
        } as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Pie and Doughnut Customization

User can draw pie and doughnut charts within the specified range using the `startAngle` and `endAngle` properties in `chartSeries`. The default value of the `startAngle` property is **0**, and the `endAngle` property is **360**. By customizing these properties, user can draw semi pie and semi doughnut charts.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public pivotData?: IDataset[];
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotViewComponent;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year' }, { name: 'Products' }],
      rows: [{ name: 'Country' }, { name: 'Quarter' }],
      formatSettings: [{ name: 'Amount', format: 'C' }],
      values: [{ name: 'Amount' }, { name: 'Sold' }]
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { chartSeries: { startAngle: 270, endAngle: 90,
type: 'Doughnut' } } as ChartSettings;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Users can get doughnut chart from pie chart and vice-versa using the `innerRadius` property in [chartSeries](#). If the property is greater than 0 percent, the doughnut chart will appear from the pie chart.

It takes the value only in percentage.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year' }, { name: 'Products' }],
            rows: [{ name: 'Country' }, { name: 'Quarter' }],
            formatSettings: [{ name: 'Amount', format: 'C' }],
            values: [{ name: 'Amount' }, { name: 'Sold' }]
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = { chartSeries: { innerRadius: '140', type:
'Pie' } } as ChartSettings;
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Exploding Series Points

Exploding can be enabled by setting the **explode** property in [chartSeries](#) to **true**. The series points will be exploded either on mouse click or touch.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year' }, { name: 'Products' }],
            rows: [{ name: 'Country' }, { name: 'Quarter' }],
            formatSettings: [{ name: 'Amount', format: 'C' }],
            values: [{ name: 'Amount' }, { name: 'Sold' }]
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
```



```

        this.chartSettings = { chartSeries: { explode: true, type: 'Pie' } }
    as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Field List

User can enable the field list by setting the property [showFieldList](#) in pivot table as **true**.

By using this, user can customize the report dynamically and view the result in pivot chart. For more information regarding the field list, refer the [field list](#) topic.

In the following sample, the **Popup** mode of field list is enabled in the pivot chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview';
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
FieldListService } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService, FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [showFieldList]='true'
[displayOption]='displayOption'></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,

```

```

        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Column' } } as
ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Grouping Bar

User can enable the grouping bar by setting the property [showGroupingBar](#) in pivot table as **true**. The grouping bar in pivot chart shows a dropdown list in value axis instead of buttons. The dropdown list holds list of value fields bounded in the [dataSourceSettings](#) and it can be switched to draw the pivot chart with the selected value field. This has been defined as the default behavior in the pivot chart component. For more information regarding the grouping bar, refer the [grouping bar](#) topic.

For multiple axis support, buttons will be placed in value axis instead of dropdown list.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='240'
[dataSourceSettings]=dataSourceSettings

```

```

    [chartSettings]='chartSettings' [showGroupingBar]='true'
    [displayOption]='displayOption'></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = { chartSeries: { type: 'Column' }} as
ChartSettings;
    }
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

For accumulation charts alone, a drop-down list will be placed in the column axis instead of the buttons. The drop-down list shows the column headers available in the pivot table. Users can dynamically switch column headers with the help of the drop-down list, and the accumulation chart will be updated accordingly.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent, GroupingBarService } from '@syncfusion/ej2-angular-
pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    providers: [PivotChartService, GroupingBarService],
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [showGroupingBar]='true'
[displayOption]='displayOption'></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public pivotData?: IDataset[];
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year' }, { name: 'Products' }],
        rows: [{ name: 'Country' }, { name: 'Quarter' }],
        formatSettings: [{ name: 'Amount', format: 'C' }],
        values: [{ name: 'Amount' }, { name: 'Sold' }]
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = { chartSeries: { type: 'Pie' } } as
ChartSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Single Axis

By default, the pivot chart will be drawn with the value field (measure) which is set first in the report under value axis. But, user can change to specific value field using the property [value](#) in [chartSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({

```

```

imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
))
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = { value: 'Amount', chartSeries: { type:
'Column' }} as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Multiple Axis

User can draw the pivot chart with multiple value fields by setting the property [enableMultipleAxis](#) in [chartSettings](#) as **true**. In the below code sample, the pivot chart will be drawn with both value fields "Sold" and "Amount" available in the [dataSourceSettings](#).

The multiple axis support is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { enableMultipleAxis: true, chartSeries: {
type: 'Column' } } as ChartSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

If the user binds more value fields, the result will be multiple pivot charts, and each chart will shrink within the parent container height. To avoid this, set the [enableScrollOnMultiAxis](#) property in

[chartSettings](#) to **true**. By doing so, each pivot chart will only shrink to a minimal "160px" - "180px" height showing a vertical scroll bar for a clear view.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }, { name: 'Products', type: 'Count'}],
      rows: [{ name: 'Country' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { enableScrollOnMultiAxis:true,
enableMultipleAxis: true, chartSeries: { type: 'Column' }} as ChartSettings;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Meanwhile, there is another way to display multiple values in a chart. In this approach, the series drawn from multiple values are grouped and displayed in a single chart. And, based on the values, multiple Y axis scales will be framed with different ranges. This can be achieved by setting the properties [enableMultipleAxis](#) as **true** and [multipleAxisMode](#) as **Single** in [chartSettings](#).

In the following code sample, the pivot chart can be seen as a single chart with multiple value fields such as **Sold** and **Amount** that are drawn as multiple Y axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
```



```

        this.chartSettings = { enableMultipleAxis: true, multipleAxisMode :
'Single', chartSeries: { type: 'Column' }} as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Additionally, to display chart series for multiple values within a single y-axis, set the properties [enableMultipleAxis](#) to **true** and the [multipleAxisMode](#) to **Combined**, in the [chartSettings](#).

The y-axis range values will be formatted using the first value field on the value axis. For example, if the first value field is in currency format and the remaining value fields are in different number formats or no format, the y-axis range values will be displayed in the currency format of the first value field.

The pivot chart in the following code sample can be seen as a single chart with multiple value fields such as **Sold** and **Amount** drawn as a single y-axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],

```

```

        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = { enableMultipleAxis: true, multipleAxisMode :
'Combined', chartSeries: { type: 'Column' } } as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show point color based on members

When multiple axes are enabled, you can display the same color for each member in the column axis by setting the [showPointColorByMembers](#) property to **true** in the [chartSettings](#). As a result, the end user can easily identify each member across different measures in the entire chart.

Furthermore, end user can see or hide specific members across different measures in the entire chart with a single click on the legend item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings

```

```

    [chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
    pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = {
        title: 'Sales Analysis', value: 'Amount', chartSeries: { type:
'Column' },
        enableMultipleAxis: true, showPointColorByMembers: true,
multipleAxisMode: 'Stacked',
        primaryYAxis: {border: {width: '0'}}
      } as any as ChartSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Series customization

User can customize series of the pivot chart using [chartSeries](#) in [chartSettings](#). The changes handled in the property will be reflected commonly in all chart series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({

```

```

imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
))
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = { chartSeries: { type: 'Column', enableTooltip:
false, border: { color: '#000', width: 2 } } } as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

User can also customize the pivot chart series individually using the [chartSeriesCreated](#) event, which occurs after the pivot chart series has been created. You can customize each series individually by iterating them.

In the following sample, the even series are hidden in the pivot chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';

```

```

import { IDataOptions, IDataset, DisplayOption, PivotChartService,
ChartSeriesCreatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'
(chartSeriesCreated)='chartSeriesCreated($event)'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = { chartSeries: { type: 'Column' } } as
ChartSettings;
    }
    chartSeriesCreated(args: ChartSeriesCreatedEventArgs) {
        for (let pos: number = 0; pos < args.series.length; pos++) {
            if (pos % 2 == 0) {
                args.series[pos].visible = false;
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Axis Customization

User can customize axis of the pivot chart using [primaryXAxis](#) and [primaryYAxis](#) properties in [chartSettings](#).

Axis customization is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

In the following sample, title of y-axis and x-axis are customized.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
      chartSeries: { type: 'Column' },
      primaryXAxis: { title: 'X axis title' },
      primaryYAxis: { title: 'Y axis title' }
```

```

    } as ChartSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

One can also customize multi-level labels of primary x-axis by using the [multiLevelLabelRender](#) event in the [chartSettings](#), which fires on rendering each multi-level label in the pivot chart. It has the following parameters:

axis - It holds the information of the current axis.

text - It allows to change the text of the multi-level label.

textStyle - It allows to customize the text font.

alignment - It allows to set the text alignment.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings'
(multiLevelLabelRender)= 'this.chartSettings?.multiLevelLabelRender'
[displayOption]='displayOption'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {

```

```

        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = {
            chartSeries: { type: 'Column' },
            multiLevelLabelRender(e) {
                e.alignment = 'Near';
                e.textStyle = { fontFamily: 'Bold', fontWeight: '400', size:
'16px', color: 'red' };
                if (e.text === ' + United Kingdom') {
                    e.text = 'Text Changed';
                    e.textStyle = { fontFamily: 'Bold', fontWeight: '800',
size: '16px', color: 'Blue' };
                }
            }
        } as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Legend customization

User can customize legend using [legendSettings](#) in [chartSettings](#). By default, legend will be visible and it can be hidden by setting the property [visible](#) in [legendSettings](#) as **false**.

The pivot chart support different types of legend shapes as follows,

- Circle
- Rectangle
- VerticalLine
- Pentagon
- InvertedTriangle
- SeriesType
- Triangle
- Diamond
- Cross
- HorizontalLine

Here **SeriesType** would act as the default shape and it can be changed using the property [LegendShape](#) in [chartSeries](#).

Also user can set the position of the legend in pivot chart using the property [position](#) in [legendSettings](#). The available options to set the legend position are as follows,

- Auto: Places the legend based on area type. This is the default.
- Top: Displays the legend at the top of the pivot chart.
- Left: Displays the legend at the left of the pivot chart.
- Bottom: Displays the legend at the bottom of the pivot chart.
- Right: Displays the legend at the right of the pivot chart.
- Custom: Displays the legend based on the given x and y values.

By default, the legend is not visible for the accumulation chart types like pie, doughnut, pyramid, and funnel.

In the below code sample, the legend shape and its position can be customized.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
        legendSettings: { position: 'Right' },
        chartSeries: { type: 'Column', legendShape: 'Pentagon' }
    } as ChartSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

User interaction

Marker and crossHair

User can enable and customize the marker and crosshair using [marker](#) and [crosshair](#) properties in [chartSettings](#) respectively.

Also user can enable and customize the crosshair tooltip for axes using [crosshairTooltip](#).

Marker and crosshair is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

In the below code sample, the marker and crosshair can be enabled and customized.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component

```

```

template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
))
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = {
            crosshair: { enable: true },
            chartSeries: {
                type: 'Line',
                marker: { fill: '#EEE', height: 10, width: 10, shape:
'Pentagon', visible: true }
            },
            primaryXAxis: { crosshairTooltip: { enable: true, fill:
'#ff0000' } },
            primaryYAxis: { crosshairTooltip: { enable: true, fill:
'#0000FF' } }
        } as any as ChartSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Zooming and panning

User can customize zooming and panning option using the property [zoomSettings](#) in [chartSettings](#).

The pivot chart support four types of zooming which can be set as follows,

- [enablePinchZooming](#)
- [enableSelectionZooming](#)
- [enableDeferredZooming](#)
- [enableMouseWheelZooming](#)

and three modes of zooming direction that specifies whether to zoom vertically or horizontally or in both ways which are,

- x: Pivot chart can be zoomed horizontally.
- y: Pivot chart can be zoomed vertically.
- x,y: Pivot chart can be zoomed both vertically and horizontally.

This can be set using the property [mode](#) in [zoomSettings](#). By default, if the pivot chart is zoomed, a toolbar would display with the options - Zoom, ZoomIn, ZoomOut, Pan, Reset. User can also customize its option using the property [toolbarItems](#) in [zoomSettings](#).

Zooming and panning is not applicable for the accumulation chart types like pie, doughnut, pyramid, and funnel.

In the below code sample, all the four types of zooming are enabled with toolbar options.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
```

```

        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
        chartSeries: {
            type: 'Column'
        },
        zoomSettings: {
            enableDeferredZooming: true,
            enableMouseWheelZooming: true,
            enablePinchZooming: true,
            enableSelectionZooming: true
        }
    } as ChartSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip

By default, tooltip for the pivot chart is enabled. User can customize it by using the property [tooltip](#) in [chartSettings](#).

The tooltip can be disabled by setting the property [enable](#) in [tooltip](#) as **false**.

In the below code sample, the default appearance of tooltip is modified.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  // specifies the template string for the pivot table component

```

```

    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
    [chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = {
        chartSeries: {
          type: 'Column'
        },
        tooltip: {
          enableMarker: true,
          textStyle: { color: '#000' },
          fill: '#FFF',
          opacity: 1,
          border: { color: '#000' }
        }
      } as ChartSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export

The pivot chart can be exported using the [chartExport](#) method which holds parameters like export type, file name, PDF orientation, width, and height in the same order. The mandatory parameters for this method are export type and file name whereas other parameters are optional.

The following are the four export types:

- PNG
- JPEG
- SVG

- PDF

In the following code sample, exporting can be done using an external button named as "Export".

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-angular-grids';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
// specifies the template string for the pivot table component
template: `<span><button ej-button
id='chartexport'>Export</button></span><div><ejs-pivotview #pivotview
id='PivotView' height='300' [dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview></div>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;
    public exportButton?: Button;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.displayOption = { view: 'Chart' } as DisplayOption;
        this.chartSettings = {
            chartSeries: {
                type: 'Column'
            }
        }
    }
}
```

```

    }
    } as ChartSettings;
    this.exportButton = new Button({ isPrimary: true });
    this.exportButton.appendTo('#chartexport');
    this.exportButton.element.onclick = (): void => {
        this.pivotGridObj?.chartExport('PNG', 'result' as
PdfExportProperties);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Print

The rendered pivot chart can be printed directly from the browser by calling [printChart](#) method.

In the following code sample, printing can be done using an external button named as "Print Chart".

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, DisplayOption, PivotChartService,
PivotViewComponent } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [PivotChartService],
// specifies the template string for the pivot table component
template: `<span><button ej-button
id='chartprint'>Print</button></span><div><ejs-pivotview #pivotview
id='PivotView' height='300' [dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'</ejs-
pivotview></div>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public chartSettings?: ChartSettings;
    public displayOption?: DisplayOption;

```



```

public exportButton?: Button;
public printButton?: Button;
@ViewChild('pivotview', {static: false})
public pivotGridObj?: PivotViewComponent;
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
        chartSeries: {
            type: 'Column'
        }
    } as ChartSettings;
    this.printButton = new Button({ isPrimary: true });
    this.printButton.appendTo('#chartprint');
    this.printButton.element.onclick = (): void => {
        this.pivotGridObj?.printChart();
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Events

[MultiLevelLabelRender](#)

The [multiLevelLabelRender](#) event triggers during the rendering of each multi-level label in the pivot chart. It allows you to customize each multi-level label's appearance and content based on your specific visualization needs. It includes the following parameters:

- [alignment](#) - It holds the text alignment for multi-level labels, as **Center, Far, or Near**.
- [axis](#) - It holds the current axis details of the multi-level labels.
- [cancel](#) - It's a boolean property that allows user to restrict the rendering of the current label.
- [customAttributes](#) - It holds the custom objects for multi-level labels.
- [text](#) - It contains the current text of the axis label.
- [textStyle](#) - It holds the font style, including font size, weight, color, and other font properties for the multi-level labels.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import {
  IDataOptions,
  IDataset,
  DisplayOption,
  PivotChartService
} from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { Observable } from 'rxjs';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PivotChartService],
  template: `<ejs-pivotview #pivotview id='PivotView' width='90%'
height='350' [dataSourceSettings]=dataSourceSettings
[chartSettings]='chartSettings' [displayOption]='displayOption'></ejs-
pivotview>`,
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  public observable = new Observable();
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }],
      rows: [{ name: 'Country' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
    };
    this.displayOption = { view: 'Chart' } as DisplayOption;
    this.chartSettings = {
      chartSeries: { type: 'Column' },
      multiLevelLabelRender: this.observable.subscribe((args: any): void => {
        // Here we customized the appearance of the labels.
        args.textStyle.size = '18px';
        args.textStyle.color = 'red';
        // Here, we have customized both the content and the position of the
        specific label, in this case, France.
        if (args.text.startsWith('France')) {
          args.text = 'Custom label';
          args.alignment = 'Far';
        }
      })
    };
  }
}

```

```

    }) as any,
  } as ChartSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Drill down in Angular Pivotview component

Drill down and drill up

The drill down and drill up action helps to view the bound data in detailed and abstract view respectively. By default, if member(s) has children, then expand and collapse icon will be displayed in the respective row/column header. On clicking the icon, expand or collapse action will be performed automatically through built-in source code. Meanwhile, leaf member(s) does not contain expand and collapse icon.

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
▼ France	450	\$714,955	526	\$1,542,104	
Mountain Bikes	197	\$335,688	238	\$405,552	
Road Bikes	253	\$379,267	288	\$1,136,552	
► Germany	440	\$563,515	496	\$1,772,104	
► United States	546	\$754,515	636	\$2,263,104	
Grand Total	1436	\$2,032,985	1658	\$5,577,312	1

Drill position

Allows to drill only the current position of the selected member and exclude the drilled data of selected member in other positions. For example, if "FY 2015" and "FY 2016" have "Quarter 1" member as child in next level, and when end user attempts to drill "Quarter 1" under "FY 2016", only it will be expanded and not "Quarter 1" under "FY 2015".

This feature is built-in and occurs every time when expand or collapse action is done for better performance.

	FY 2015		FY 2016		
	► Q1	FY 2015 Tot...	▼ Q1	Road Bikes	Q1 Total
France	114	114	27	67	
Germany	74	74	97	57	
United States	144	144	57	97	
Grand Total	332	332	181	221	

Expand All

This property is applicable only for the relational data source.

Allows to either expand or collapse all headers that are displayed in row and column axes. To display all headers in expanded state, set the property [expandAll](#) to **true** and to collapse all headers, set the property [expandAll](#) to **false**. By default, [expandAll](#) property is set to **false**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Expand all headers for specific fields

This property is applicable only for the relational data source.

Allows to expand or collapse all headers for specific fields (only) in row and column axes. To expand headers for a specific field in row or column axis, set the property [expandAll](#) in [rows](#) or [columns](#) to **true**. By default, [expandAll](#) property in [rows](#) or [columns](#) is set to **false**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year', expandAll:
true }, { name: 'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' , expandAll: true }, { name: 'Products'
}],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Expand all except specific member(s)

This option is applicable only for the relational data source.

In addition to the previous topic, there is an enhancement to expand all headers except specific header(s) and similarly to collapse all headers except specific header(s). To achieve this, [drilledMember](#) is used. The required properties of the [drilledMember](#) are explained below:

- [name](#): It allows to set the field name whose member(s) needs to be specifically drilled.
- [items](#): It allows to set the exact member(s) which needs to be drilled.

The [drilledMember](#) option always works in vice-versa with respect to the property [expandAll](#) in pivot table. For example, if [expandAll](#) is set to **true**, then the member(s) added in [items](#) collection alone will be in collapsed state.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Expand specific member(s)

End user can also manually expand or collapse specific member(s) in each fields under row and column axes using the [drilledMembers](#) from code behind. The required properties of the [drilledMembers](#) are explained below:

- [name](#): It allows to set the field name whose member(s) needs to be specifically drilled.
- [items](#): It allows to set the exact member(s) which needs to be drilled.
- [delimiter](#): It allows to separate next level of member from its parent member.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}
```

```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Event*Drill*

The event [drill](#) triggers every time when a field is expanded or collapsed. For instance using this event user can alter delimiter and drill action for the respective item. It has the following parameters:

- **drillInfo** - It holds the current drilled item information.
- **pivotview** - It holds pivot table instance.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, DrillArgs } from '@syncfusion/ej2-angular-
pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
(drill)='drill($event)'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  drill(args: DrillArgs) {
    //args.drillInfo -> get current drilled information
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```



```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionBegin

The event [actionBegin](#) triggers when the UI actions such as drill down and drill up begin. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. The following are the UI actions and their names:

Action	Action Name
Expand	Drill down
Collapse	Drill up

- **cancel**: It allows user to restrict the current action.

In the below sample, drill down and drill up action can be restricted by setting the **args.cancel** option to **true** in the [actionBegin](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotActionBeginEventArgs } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,

```

```

        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings (actionBegin)='actionBegin($event)'
width=width></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    actionBegin(args: PivotActionBeginEventArgs): void {
      if (args.actionName == 'Drill down' || args.actionName == 'Drill
up') {
        args.cancel = true;
      }
    }
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.width = '100%';
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionComplete

The event [actionComplete](#) triggers when a UI action such as drill down or drill up, is completed. This allows user to identify the current UI actions being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action completed. The following are the UI actions and their names:

Action	Action Name
--------	-------------

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

| [Expand](#) | Drill down |

| [Collapse](#) | Drill up |

- **actionInfo**: It holds the unique information about the current UI action. For example, if drill down action is completed, the event argument contains information such as field name and the drill information.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotActionCompleteEventArgs } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
(actionComplete)=actionComplete($event) width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  actionComplete(args: PivotActionCompleteEventArgs): void {
    if (args.actionName == 'Drill down' || args.actionName == 'Drill
up') {

      // Triggers when the drill operations are completed.
    }
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. The following are the UI actions and their names:

| Action | Action Name |

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

| [Expand](#) | Drill down |

| [Collapse](#) | Drill up |

- **errorInfo**: It holds the error information of the current UI action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotActionFailureEventArgs } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
(actionFailure)= 'actionFailure($event)' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  actionFailure(args: PivotActionFailureEventArgs): void {
```

```

        if (args.actionName == 'Drill down' || args.actionName == 'Drill
up') {
            // Triggers when the current UI action fails to achieve the
desired result.
        }
    }
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.width = '100%';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Data Shaping

Aggregation in Angular Pivotview component

This feature is applicable only for the relational data source.

End user can perform calculations over a group of values (exclusively for value fields bound in value axis) using the aggregation option. By default, values are added (summed) together. The other aggregation types are explained below.

The fields with data type such as number support all aggregation types mentioned below except for **“CalculatedField”**. The fields with data type such as string, date, datetime, boolean, etc., support **“Count”** and **“DistinctCount”** aggregation types alone.

Operator	Description
Sum	Displays the pivot table values with sum.
Product	Displays the pivot table values with product.
Count	Displays the pivot table values with count.
DistinctCount	Displays the pivot table values with distinct count.
Min	Displays the pivot table with minimum value.

| Max| Displays the pivot table with maximum value. |

| Avg| Displays the pivot table values with average. |

| Median| Displays the pivot table values with median. |

| Index| Displays the pivot table values with index. |

| PopulationStDev| Displays the pivot table values with standard deviation of population. |

| SampleStDev| Displays the pivot table values with sample standard deviation. |

| PopulationVar| Displays the pivot table values with variance of population. |

| SampleVar| Displays the pivot table values with sample variance. |

| RunningTotals| Displays the pivot table values with running totals. |

| DifferenceFrom| Displays the pivot table values with difference from the value of the base item in the base field. |

| PercentageOfDifferenceFrom| Displays the pivot table values with percentage difference from the value of the base item in the base field. |

| PercentageOfGrandTotal| Displays the pivot table values with percentage of grand total of all values. |

| PercentageOfColumnTotal| Displays the pivot table values in each column with percentage of total values for the column. |

| PercentageOfRowTotal| Displays the pivot table values in each row with percentage of total values for the row. |

| PercentageOfParentTotal| Displays the pivot table values with percentage of total of all values based on selected field. |

| PercentageOfParentColumnTotal| Displays the pivot table values with percentage of its parent total in each column. |

| PercentageOfParentRowTotal| Displays the pivot table values with percentage of its parent total in each row. |

| CalculatedField| Displays the pivot table with calculated field values. It allows user to create a new calculated field alone. |

Assigning aggregation type for value fields through API

For each value field, the aggregation type can be set using the property [type](#) in [values](#). Meanwhile, aggregation types like **DifferenceFrom** and **PercentageOfDifferenceFrom** can check for specific field of specific item using [baseField](#) and [baseItem](#) properties. Likewise, **PercentageOfParentTotal** type can for specific field using [baseField](#) property. For instance, the aggregation type **DifferenceFrom** would intake the specified field and its corresponding member as input and its value is compared across other members in the same field and also across different fields to formulate an appropriate output value.

- [type](#): It allows to set the aggregate type of the field.
- [baseField](#): It allows to set the specific field to aggregate the values.
- [baseItem](#): It allows to set the specific member to aggregate the values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' } ],
      values: [{ name: 'Sold', caption: 'Units Sold',
type: 'DifferenceFrom', baseField: 'Country', baseItem: 'France' }, { name:
'Amount', caption: 'Sold Amount', type: 'Sum' } ],
      rows: [{ name: 'Country' }, { name: 'Products' } ],
      formatSettings: [{ name: 'Amount', format: 'C0' } ],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

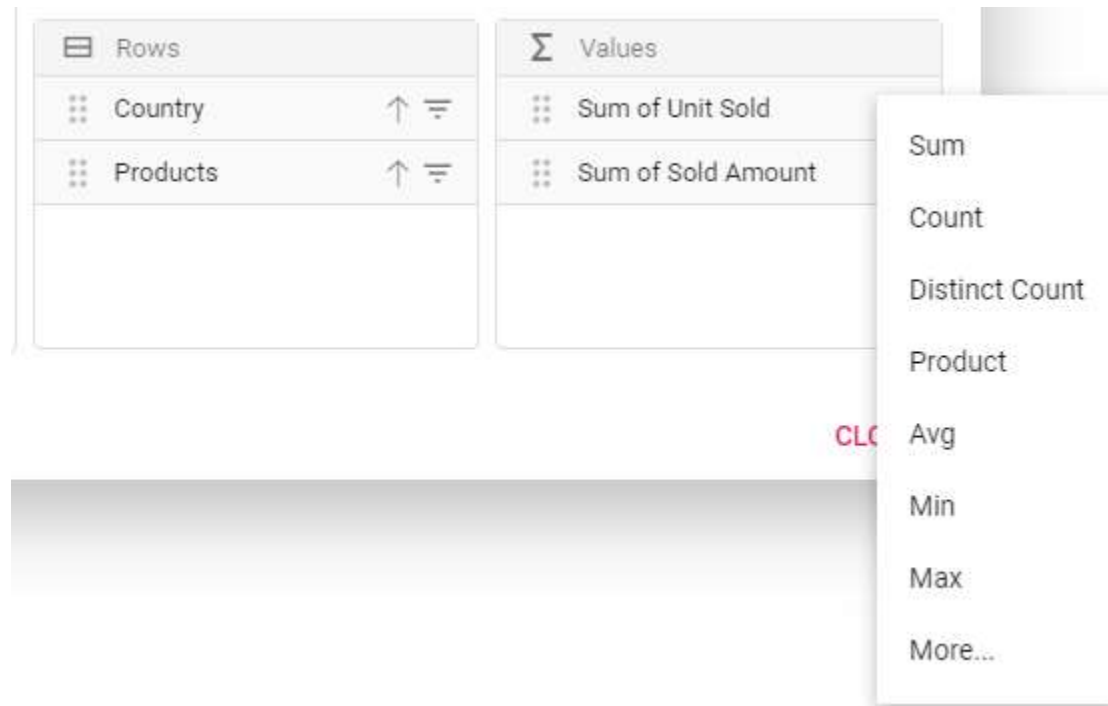
```

By default, the aggregation will be considered as **Sum** to the value fields which had number type and for the value fields which had non-number type values such as string, date, datetime, boolean, etc., the aggregation type will be considered as **Count**.

Modifying aggregation type for value fields at runtime

Aggregation types can be changed easily through UI at runtime. The value fields bound to grouping bar and field list appears with a dropdown icon which helps to select an appropriate aggregation type for the respective value field. On selection, the values in the pivot table will be changed dynamically.

<!-- markdownlint-disable MD012 -->



Sum of Units Sold	Sum	Drop filter here	
Sum of Sold Amount	Count	Year ↑ = ×	Quarter ↑ = ×
Country ↑	Distinct Count	► FY 2015	► FY 2016
Products	Product	Units Sold	Sold Amount
► France	Min	450	\$714,955
► Germany	Max	440	\$563,515
► United States	Avg	546	\$754,515
Grand Total	More...	1436	\$2,032,985

Show desired aggregation types in its dropdown menu

By default, all the aggregation types are displayed in the dropdown menu available in buttons. However, based on the request for an application, we may need to show selective aggregation types on our own. This can be achieved using the [aggregateTypes](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```



```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, GroupingBarService, AggregateTypes, IDataset } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width showGroupingBar='true'
[aggregateTypes]='aggregateTypesOption'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  public aggregateTypesOption?: AggregateTypes[];
  ngOnInit(): void {
    this.width = '100%';
    this.aggregateTypesOption = ['DistinctCount', 'Avg', 'Product'],
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount', type: 'Sum' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      filters: [],
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hiding aggregation type from button text

By default, in value axis each field would be displayed by its name and aggregation type together. To hide aggregation type and display field name alone, set the property [showAggregationOnValueField](#) in [dataSourceSettings](#) to **false**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

],
standalone: true,
selector: 'app-container',
providers: [GroupingBarService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showGroupingBar='true'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            showAggregationOnValueField: false
        };
        this.width = "100%";
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hiding aggregation type icon from UI

By default, the icon to set aggregation type is enabled in the grouping bar. To disable this icon, set the property [showValueTypelcon](#) in [groupingBarSettings](#) to **false**.

Icon to change the aggregation type can be hidden only in Grouping Bar but not in Field List at the moment.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService,
GroupingBarSettings } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [GroupingBarService],
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showGroupingBar='true'
[groupingBarSettings]='groupingSettings'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public groupingSettings?: GroupingBarSettings;
    public width?: string;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            showAggregationOnValueField: false
        };
        this.width = "100%";
        this.groupingSettings = {
            showValueTypeIcon: false
        } as GroupingBarSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Event

AggregateCellInfo

The event [aggregateCellInfo](#) triggers every time while rendering value cell. This allows user to change the cell value and skip formatting if applied. It has following parameters:

- **fieldName** - It holds current cell's field name.
- **row** - It holds current cell's row value.
- **column** - It holds current cell's row value.
- **value** - It holds value of current cell.
- **cellSets** - It holds raw data for the aggregated value cell.
- **rowCellType** - It holds row cell type value.
- **columnCellType** - It holds column cell type value.
- **aggregateType** - It holds aggregate type of the cell.
- **skipFormatting** - boolean property, it allows to skip formatting if applied.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, GroupingBarService, AggregateEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showGroupingBar='true'
(aggregateCellInfo)='aggregateCell($event)'></ejs-pivotview>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  aggregateCell(args: AggregateEventArgs) {
    args.skipFormatting = true;
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],

```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        showAggregationOnValueField: false
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionBegin

The event [actionBegin](#) triggers when clicking and selecting the aggregate type via the dropdown icon in the value field button, which is present in both grouping bar and field list UI. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. For example, while performing aggregation, the action name will be shown as **Aggregate field**.
- **fieldInfo**: It holds the selected value field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

- **cancel**: It allows user to restrict the current action.

In the following example, action taken during aggregation type selection via dropdown icon can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, GroupingBarService, FieldListService,
PivotActionBeginEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    providers: [GroupingBarService, FieldListService],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showGroupingBar='true'
showFieldList='true' (actionBegin)= 'actionBegin($event)' width=width></ejs-
pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    actionBegin(args: PivotActionBeginEventArgs): void {
      if (args.actionName == 'Aggregate field') {
        args.cancel = true;
      }
    }
    ngOnInit(): void {
      this.width = "100%";
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataSet[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' , showFilterIcon:
false }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionComplete

The event [actionComplete](#) triggers when a UI action, such as applying aggregation using the dropdown icon via the value field button, which is present in both the grouping bar and the field list UI, is completed. This allows user to identify the current UI action being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action completed. For example, after completing the aggregation, the action name will be shown as **Field aggregated**.
- **fieldInfo**: It holds the selected value field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, GroupingBarService, FieldListService,
 PivotActionCompleteEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService, FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showGroupingBar='true'
(actionComplete)=actionComplete($event) showFieldList='true'
width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  actionComplete(args: PivotActionCompleteEventArgs): void {
    if (args.actionName == 'Field aggregated') {
      // Triggers when the aggregation type is applied.
    }
  }
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' , showFilterIcon:
false }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. For example, if the action fails while performing the aggregation, then the action name will be shown as **Aggregate field**.
- **errorInfo**: It holds the error information of the current UI action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, GroupingBarService, FieldListService,
 PivotActionFailureEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [GroupingBarService, FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showGroupingBar='true'
(actionFailure)='actionFailure($event)' showFieldList='true'
width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  actionFailure(args: PivotActionFailureEventArgs): void {
    if (args.actionName == 'Aggregate field') {
      // Triggers when the current UI action fails to achieve the
desired result.
    }
  }
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```



```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' , showFilterIcon:
false }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Calculated field in Angular Pivotview component

Allows end user to create a new calculated field in the pivot table, based on available fields from the bound data source or using simple formula with basic arithmetic operators. It can be added at runtime through the built-in dialog, invoked from Field List UI. To do so, set the [allowCalculatedField](#) property to **true** in the pivot table. End user can now see a "CALCULATED FIELD" button enabled in Field List UI automatically, which on clicking will invoke the calculated field dialog and perform necessary operation.

Calculated field can also be included in the pivot table through code behind using the [calculatedFieldsSettings](#) . The required properties to create a new calculate field are:

- [name](#): It allows to indicate the calculated field with a unique name.
- [formula](#): It allows to set the formula.
- [formatSettings](#): It helps to set the number format for the resultant value.

To use calculated field option, you need to inject the `CalculatedFieldService` module in pivot table.

The calculated field is applicable only for value fields. Also, calculated field created through code behind will be automatically listed in the UI dialog as well.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, CalculatedFieldService, PivotView,
FieldListService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,

```

```

    selector: 'app-container',
    providers: [FieldListService, CalculatedFieldService],
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true'
    allowCalculatedField='true' width=width></ejs-pivotview></div>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public width?: string;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }, { name: 'Total', caption: 'Total
Amount', type: 'CalculatedField' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }, { name:
'Total', format: 'C2' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
      };
      this.width = '100%';
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Meanwhile, user can also view calculated field dialog in UI by invoking `createCalculatedFieldDialog` method on an external button click which is shown in the below code sample.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, CalculatedFieldService, PivotView, IDataset } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';

```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [CalculatedFieldService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowCalculatedField='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='calculatedfield'>Calculated
Field</button></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public width?: string;
  @ViewChild('pivotview',{static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }, { name: 'Total', caption: 'Total
Amount', type: 'CalculatedField' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }, { name:
'Total', format: 'C2' }],
      filters: [],
      calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#calculatedfield');
    this.button.element.onclick = (): void => {

this.pivotGridObj?.calculatedFieldModule.createCalculatedFieldDialog();
    };
  }
}

```

MAIN.TS

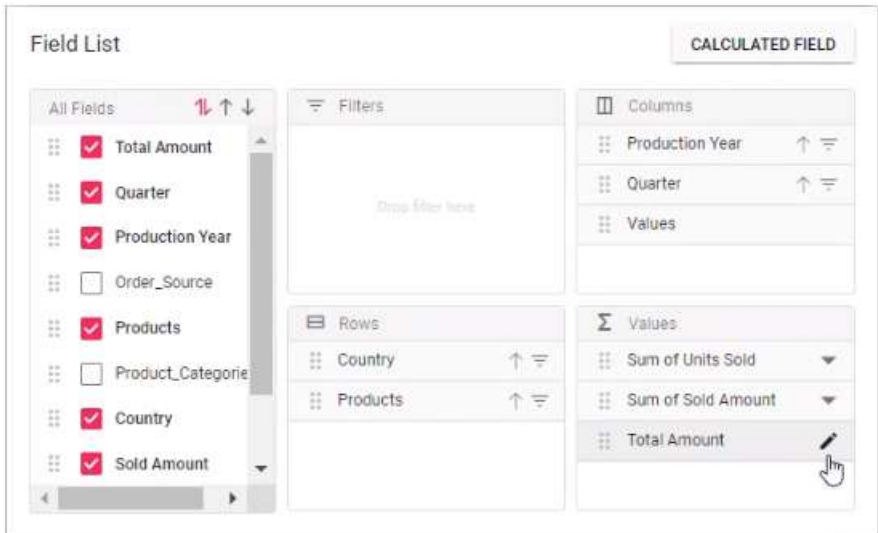
```

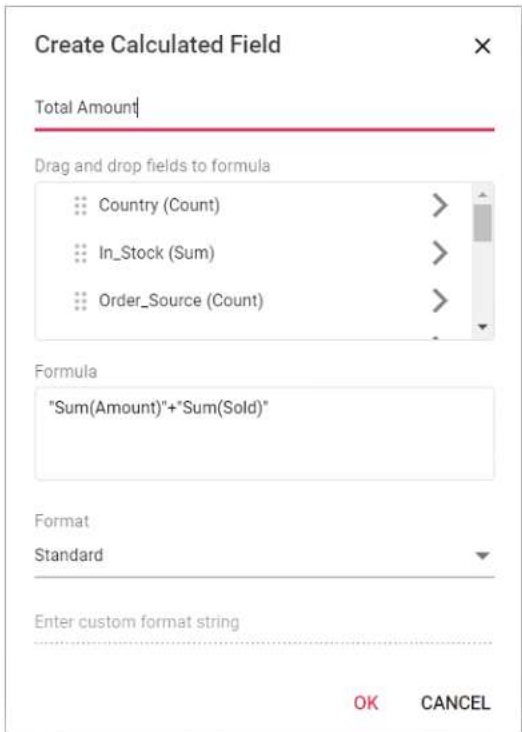
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Editing through the field list and the grouping bar

User can also modify the existing calculated field using the built-in edit option available directly in the field list (or) grouping bar. To do so, click the "Edit" icon available in the calculated field button. Now the calculated field dialog is opened and the current calculated field name, formula and format can be changed at runtime.





Renaming the existing calculated field

Existing calculated field can be renamed only through the UI at runtime. To do so, open the calculated field dialog, select the target field and click "Edit" icon. User can now see the existing name getting displayed in the text box at the top of the dialog. Now, change the name based on user requirement and click "OK".

<!-- markdownlint-disable MD012 -->

Create Calculated Field ×

Enter the field name

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field) **Edit**
- Units Sold (Sum)

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)" * 250

Format

None

Enter custom format string

OK CANCEL

Create Calculated Field

×

Total Sales Amount

Drag and drop fields to formula

Sold Amount (Sum)

>

Total Amount (Calculated Field)

✕

✎

Units Sold (Sum)

>

Formula

"Sum(Amount)"+"Sum(Sold)"

Format

Standard

Enter custom format string

OK

CANCEL

Editing the existing calculated field formula

Existing calculated field formula can be edited only through the UI at runtime. To do so, open the calculated field dialog, select the target field and click "Edit" icon. User can now see the existing formula getting displayed in a multiline text box at the bottom of the dialog. Now, change the formula based on user requirement and click "OK".

Create Calculated Field

×

Enter the field name

Drag and drop fields to formula

Sold Amount (Sum)

>

Total Amount (Calculated Field)

✕

✎

Units Sold (Sum)

>

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Format

None

Enter custom format string

OK

CANCEL

The screenshot shows the 'Create Calculated Field' dialog box. At the top, the title is 'Create Calculated Field' with a close button (X). Below the title, the text 'Total Amount' is displayed. The main section is titled 'Drag and drop fields to formula'. It contains a list of fields: 'Sold Amount (Sum)', 'Total Amount (Calculated Field)', and 'Units Sold (Sum)'. The 'Total Amount (Calculated Field)' is highlighted, and a hand icon is shown dragging it. Below the list, the 'Formula' section contains the text '*Sum(Amount)*'+*Sum(Sold)*'. The 'Format' section shows a dropdown menu set to 'Standard'. At the bottom, there is a text input field for 'Enter custom format string' and two buttons: 'OK' and 'CANCEL'.

Reusing the existing formula in a new calculate field

While creating a new calculated field, if user wants to add the formula of an existing calculated field, it can be done easily. To do so, simply drag-and-drop the existing calculated field to the "Formula" section.

Create Calculated Field

Sales Amount

Drag and drop fields to formula

Sold Amount (Sum)

Total Amount (Calculated Field)

Total Amount (Calculated Field)

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Format

None

Enter custom format string

OK

CANCEL

Create Calculated Field

Sales Amount

Drag and drop fields to formula

Sold Amount (Sum)

Total Amount (Calculated Field)

Units Sold (Sum)

Formula

Example: ("Sum(Order_Count)" + "Sum(In_Stock)") * 250

Format

None

Enter custom format string

OK

CANCEL

Copyright © 2001 -2024 Syncfusion Inc.

236

Create Calculated Field

Sales Amount

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field)
- Units Sold (Sum)

Formula

Sum(Amount)+*Sum(Sold)*

Format

None

Enter custom format string

OK CANCEL

Apply the format to the calculated field values

Values in a new or existing calculated field can be formatted via the calculated field UI or code behind. The [formatSettings](#) property in code-behind can be used to specify the desired format. For more information about the supported formats refer [here](#).

To apply format to calculated field values at runtime via UI, a built-in dropdown under the "Format" label is available, from which the user can select the pre-defined format options listed below.

- **Standard** - Denotes the numeric type.
- **Currency** - Denotes the currency type.
- **Percent** - Denotes the percentage type.
- **Custom** - Denotes the custom format. For example: "C2". This shows the value "9584.3" as "\$9584.30."
- **None** - Denotes that no format will be applied.

By default, **None** will be selected from the dropdown.

Create Calculated Field [X]

Total Amount

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field)**
- Units Sold (Sum)

Formula

"Sum(Amount)*Sum(Sold)"

Format

Standard

- Standard**
- Currency
- Percent
- Custom
- None

In addition, you can specify the desired custom formats by selecting the **Custom** option from the "Format" dropdown.

Create Calculated Field [X]

Sales Amount

Drag and drop fields to formula

- Sold Amount (Sum)
- Total Amount (Calculated Field)
- Units Sold (Sum)

Formula

"Sum(Amount)*Sum(Sold)"

Format

Custom

C2

OK CANCEL

Supported operators and functions for the calculated field formula

Below is a list of operators and functions that can be used in the formula to create the calculated fields.

- `+` – addition operator.

```
`typescript
```

```
Syntax: X + Y
```

```
,
```

- `-` – subtraction operator.

```
`typescript
```

```
Syntax: X - Y
```

```
,
```

- `*` – multiplication operator.

```
`typescript
```

```
Syntax: X * Y
```

```
,
```

- `/` – division operator.

```
`typescript
```

```
Syntax: X / Y
```

```
,
```

- `^` – power operator.

Grouping in Angular Pivotview component

This feature is applicable only for the relational data source.

Grouping is the most-useful feature in pivot table and the component automatically groups date, time, number and string. For example, the date type can be formatted and displayed based on year, quarter, month, and more. Likewise, the number type can be grouped range-wise, such as 1-5, 6-10, etc. These group fields will act as individual fields and allows users to drag them between different axes such as columns, rows, values, and filters and create pivot table at runtime.

```
<!-- markdownlint-disable MD012 -->
```

Filtering in Angular Pivotview component

Filtering allows to view the pivot table with selective records based on members that can be either included or excluded through UI and code-behind.

The following are the three different types of filtering:

- Member filtering
- Label filtering

- Value filtering

When all the above filtering options are disabled via code-behind, then the filter icon would be disabled in the field list or grouping bar UI.

Member filtering

Sorting in Angular Pivotview component

Member Sorting

Allows to order field members in rows and columns either in ascending or descending order. By default, field members in rows and columns are in ascending order.

Drill through in Angular Pivotview component

Editing in Angular Pivotview component

This feature is applicable only for the relational data source.

Data Formatting

Number formatting in Angular Pivotview component

Allows you to specify the required display format that should be used in values of the pivot table.

Supported display formats are:

- Number
- Currency
- Percentage
- Custom

Conditional formatting in Angular Pivotview component

Allows end user to change the appearance of the pivot table value cells with its background color, font color, font family, and font size based on specific conditions.

Report Manipulation

Grouping bar in Angular Pivotview component

Field list in Angular Pivotview component

The pivot table provides a built-in Field List similar to Microsoft Excel. It allows to add or remove fields and also rearrange them between different axes, including column, row, value, and filter along with sort and filter options dynamically at runtime.

The field list can be displayed in two different formats to interact with pivot table. They are:

- **In-built Field List (Popup):** To display the field list icon in pivot table UI to invoke the built-in dialog.
- **Stand-alone Field List (Fixed):** To display the field list in a static position within a web page.

In-built Field List (Popup)

To enable the field list in pivot table UI, set the [showFieldList](#) property in pivot view to **true**. A small icon will appear on the top left corner of the pivot table and clicking on this icon, field list dialog will appear.

The field list icon will be displayed at the top right corner of the pivot table, when grouping bar is enabled.

To use field list, you need to inject the `FieldListService` module in pivot table.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
 '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
width=width></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      allowLabelFilter: true,
      allowValueFilter: true,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Stand-alone Field List (Fixed)

The field list can be rendered in a static position, anywhere in web page layout, like a separate component. To do so, you need to set [renderMode](#) property to **Fixed** in pivot fieldlist.

To make field list interact with pivot table, you need to use the **updateView** and **update** methods for data source update in both field list and pivot table simultaneously.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
    EnginePopulatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
styleUrls: ['./app.component.css'],
// specifies the template string for the pivot table component
template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Fixed"
(enginePopulated)='afterPopulate($event)' allowCalculatedField='true'
(load)='onLoad()' (dataBound)='ondataBound()'></ejs-pivotfieldlist>
    <ejs-pivotview #pivotview id='PivotViewFieldList' width='99%' height='530'
(enginePopulated)='afterEnginePopulate($event)'
[gridSettings]='gridSettings'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldlistObj?: PivotFieldListComponent;
    afterPopulate(arge: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.updateView(this.pivotGridObj);
        }
    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.update(this.pivotGridObj);
        }
    }
}
```

```

onLoad(): void {
    if (Browser.isDevice) {
        (this.fieldlistObj as PivotFieldListComponent).renderMode =
'Popup';
        (this.fieldlistObj as PivotFieldListComponent).target =
'.control-section';
        (document.getElementById('PivotFieldList') as
HTMLElement).removeAttribute('style');
        setStyleAttribute(document.getElementById('PivotFieldList') as
HTMLElement, {
            'height': 0,
            'float': 'left'
        });
    }
}
ondataBound(): void {
    if (Browser.isDevice) {
        prepend([document.getElementById('PivotFieldList') as
HTMLElement], document.getElementById('PivotView') as HTMLElement);
    }
}
ngOnInit(): void {
    this.gridSettings = {
        columnWidth: 140
    } as GridSettings;
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Search desired field

End user can search for desired field in the field list UI by typing the field name into the search box at runtime. It can be enabled by setting the [enableFieldSearching](#) property to **true** via code-behind.

By default, field search option is disabled in the field list UI.

To enable search box in the static field list UI, set the [enableFieldSearching](#) property to **true** in `ejs-pivotfieldlist`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
    EnginePopulatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
styleUrls: ['./app.component.css'],
// specifies the template string for the pivot table component
template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Fixed"
(enginePopulated)='afterPopulate($event)' allowCalculatedField='true'
enableFieldSearching='true' (load)='onLoad()'
(dataBound)='ondataBound()'></ejs-pivotfieldlist>
    <ejs-pivotview #pivotview id='PivotViewFieldList' width='99%' height='530'
(enginePopulated)='afterEnginePopulate($event)'
[gridSettings]='gridSettings'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldlistObj?: PivotFieldListComponent;
    afterPopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.updateView(this.pivotGridObj);
        }
    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            (this.fieldlistObj as
PivotFieldListComponent).update(this.pivotGridObj);
        }
    }
    onLoad(): void {
        if (Browser.isDevice) {
            (this.fieldlistObj as PivotFieldListComponent).renderMode =
'Popup';

```



```

        (this.fieldlistObj as PivotFieldListComponent).target =
        '.control-section';
        (document.getElementById('PivotFieldList') as
        HTMLElement).removeAttribute('style');
        setStyleAttribute(document.getElementById('PivotFieldList') as
        HTMLElement, {
            'height': 0,
            'float': 'left'
        });
    });
}
}
ondataBound(): void {
    if (Browser.isDevice) {
        prepend([document.getElementById('PivotFieldList') as
        HTMLElement], document.getElementById('PivotView') as HTMLElement);
    }
}
ngOnInit(): void {
    this.gridSettings = {
        columnWidth: 140
    } as GridSettings;
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
        'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
        'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

To enable search box in the pivot table's built-in popup field list UI, set the [enableFieldSearching](#) property to **true** in `ejs-pivotview`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';

```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
enableFieldSearching='true' width=width></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      allowLabelFilter: true,
      allowValueFilter: true,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

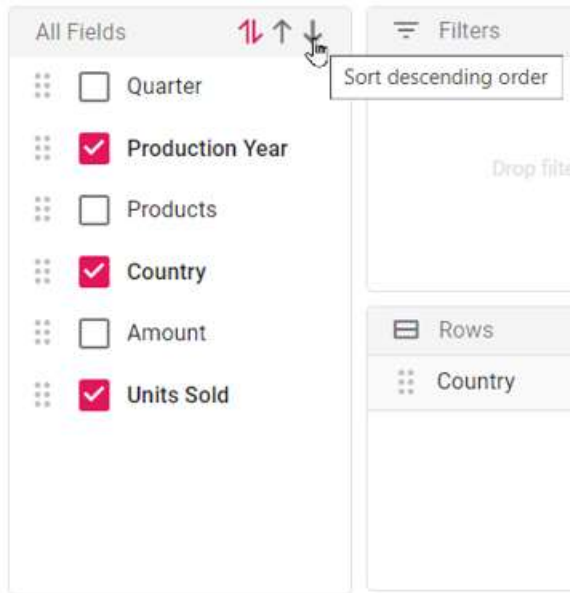
```

Option to sort fields

End user can sort fields in the field list UI to ascending (or) descending (or) default order (as obtained from the data source) using the built-in sort icons.

By default, fields are displayed in the default order.

Field List



Sort fields in a desired order

To display the fields in descending order by default, set the [defaultFieldListOrder](#) property to **Descending** in the [load](#) event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, GroupingBarService, FieldListService,
LoadEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
showGroupingBar='true' (load)='onLoad($event)' width=width></ejs-
pivotview></div>`
})
export class AppComponent {
  public width?: string;
```

```

public dataSourceSettings?: IDataOptions;
onLoad(args: LoadEventArgs): void {
    args.defaultFieldListOrder = 'Descending';
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        columns: [{ name: 'Year', caption: 'Production Year' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        rows: [{ name: 'Country' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Group fields under desired folder name

In the field list UI, you can display fields by grouping them under the desired folder name. It can only be configured via code-behind by setting the [groupName](#) property in [fieldMapping](#).

You can only group fields to one level using the [groupName](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview';
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService, GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [
        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService, GroupingBarService],
    // specifies the template string for the pivot table component
    template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
showGroupingBar='true' width=width></ejs-pivotview></div>`
})

```

```

    })
    export class AppComponent {
        public width?: string;
        public dataSourceSettings?: IDataOptions;
        ngOnInit(): void {
            this.width = '100%';
            this.dataSourceSettings = {
                dataSource: Pivot_Data as IDataset[],
                expandAll: false,
                allowLabelFilter: true,
                allowValueFilter: true,
                columns: [{ name: 'Year', caption: 'Production Year' }],
                values: [{ name: 'Sold', caption: 'Units Sold' }],
                rows: [{ name: 'Country' }],
                formatSettings: [{ name: 'Amount', format: 'C0' }],
                filters: [],
                fieldMapping: [
                    { name: 'Quarter', groupName: 'Product category' },
                    { name: 'Products', groupName: 'Product category' },
                    { name: 'Amount', groupName: 'Product category', caption:
'Sold Amount' },
                ]
            };
        }
    }
}

```

MAIN.TS

```

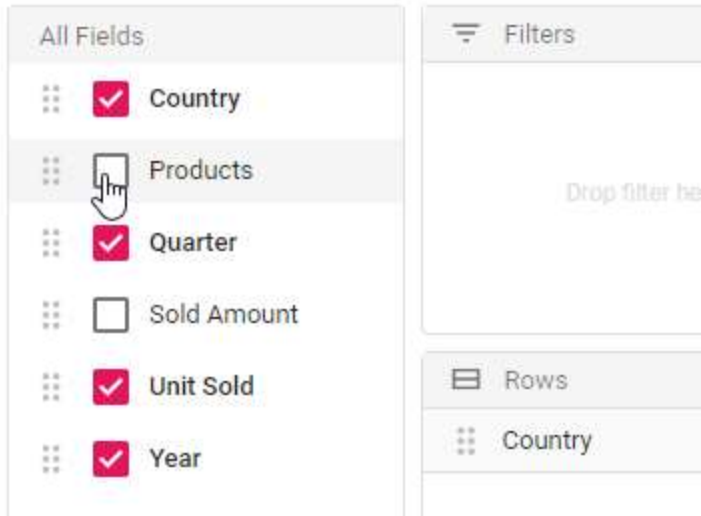
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Add or remove fields

Using check box besides each field, end user can select or unselect to add or remove fields respectively from the report at runtime.

Field List



Remove specific field(s) from displaying

When a data source is bound to the component, fields will be automatically populated inside the Field List. In such case, user can also restrict specific field(s) from displaying. To do so, set the appropriate field name(s) in [excludeFields](#) property belonging to [dataSourceSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
width=width></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: [],
        excludeFields: ["Products", "Amount" ]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Re-arranging fields

In-order to re-arrange, drag any field from the field list and drop it into the column, row, value, or filter axis using the drag-and-drop holder. It helps end user to alter the report at runtime.

Filters	Columns
<div> <div></div> <div>Products (All)</div> <div></div> </div>	<div> <div></div> <div>Year</div> <div>↑</div> <div></div> </div>
	<div> <div></div> <div>Quarter</div> <div>↑</div> <div></div> </div>
Rows	Values
<div> <div></div> <div>Country</div> <div>↑</div> <div></div> </div>	<div> <div></div> <div>Sum of Unit Sold</div> <div>▼</div> </div>
	<div> <div></div> <div>Sum of Sold Amount</div> <div>▼</div> </div>

Filtering members

Using the filter icon besides each field in row, column and filter axes, members can be either included or excluded at runtime. To know more about member filtering, [refer](#) here.

Filters

Drop filter here

Columns

Year

Quarter

Rows

Country

Products

Values

Sum of Unit Sold

Sum of Sold Amount

Year

Search "Year"

☐ All

☐ FY 2015

☐ FY 2016

☐ FY 2017

☒ FY 2018

OK

CANCEL

	FY 2018		Grand Total	
	Units Sold	Sold Amount	Units Sold	Sold Amount
France	16	\$27,264	16	\$27,264
Germany	96	\$77,264	96	\$77,264
United States	76	\$97,264	76	\$97,264
Grand Total	188	\$201,792	188	\$201,792

Sorting members

Using the sort icon besides each field in row and column axes, members can be arranged either in ascending or descending order at runtime. To know more about member sorting, [refer](#) here.

Filters

Drop filter here

Columns

- Year ↑
- Quarter ↑

Rows

- Country ↑
- Products ↑

Values

- Sum of Unit Sold ▼
- Sum of Sold Amount ▼

	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
United States	546	\$754,515	636	\$2,263,104	
Germany	440	\$563,515	496	\$1,772,104	
France	450	\$714,955	526	\$1,542,104	
Grand Total	1436	\$2,032,985	1658	\$5,577,312	1

Calculated fields

The calculated field support allows end user to add a new calculated field based on the available fields from the bound data source using basic arithmetic operators. To enable this support in Field List UI, set the [allowCalculatedField](#) property in pivot table to **true** in pivot table. Now a button will be seen automatically inside the field list UI which will invoke the calculated field dialog on click. To know more about calculated field, [refer](#) here.

Field List

All Fields

Country

Products

Quarter

Sold Amount

Total Amount

Unit Sold

Year

Filters

Drop filter here

Rows

Country

Products

Columns

Year

Quarter

VALUES

Sum of Unit Sold

Sum of Sold Amount

Total Amount

CALCULATED FIELD

CLOSE

Copyright © 2001 -2024 Syncfusion Inc.

254

Create Calculated Field

×

Total Amount

Drag and drop fields to formula

Total Amount (CalculatedField)

Unit Sold (Sum)

Year (Count)

Formula

"Sum(Amount)*"Sum(Sold)"

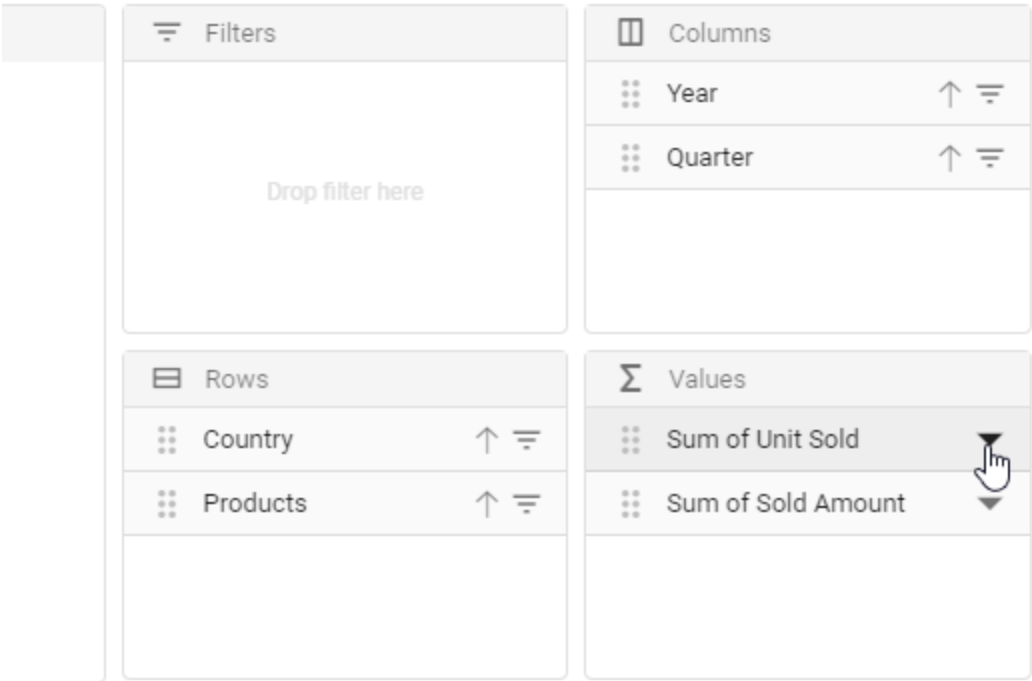
OK

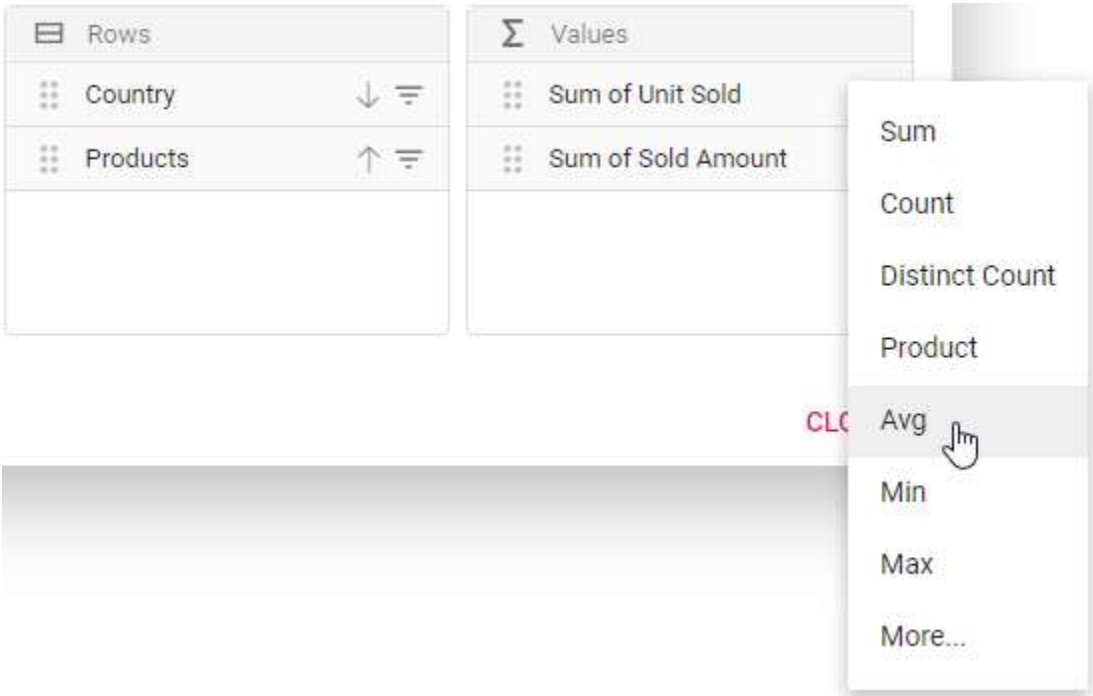
CANCEL

	FY 2015			FY 2016	
	Units Sold	Sold Amount	Total Amount	Units Sold	Sold Amount
France	450	\$714,955	\$715,405	526	\$1,542,
Germany	440	\$563,515	\$563,955	496	\$1,772,
United States	546	\$754,515	\$755,061	636	\$2,263,
Grand Total	1436	\$2,032,985	\$2,034,421	1658	\$5,577,

Changing aggregation type of value fields at runtime

End user can perform calculations over a group of values using the aggregation option. The value fields bound to the field list, appears with a dropdown icon, helps to select an appropriate aggregation type at runtime. On selection, the values in the Pivot Table will be changed dynamically. To know more about aggregation, [refer](#) here.





	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
United States	68.25	\$754,515	79.5	\$2,263,104	91.5
Germany	55	\$563,515	62	\$1,772,104	70.5
France	56.25	\$714,955	65.75	\$1,542,104	76.5
Grand Total	59.83333333...	\$2,032,985	69.08333333...	\$5,577,312	70.5

Defer layout update

Defer layout update support to update the pivot table only on demand and not during every user action. To enable this support in Field List UI, set the [allowDeferLayoutUpdate](#) property to **true** in pivot table. Now a check box inside Field List UI will be seen in checked state, allowing pivot table to update only on demand. To know more about defer layout, [refer](#) here.

Unit Sold

Year

Rows

Country

Products

Values

Sum of Unit Sold

Sum of Sold Amount

☒ Defer Layout Update
 APPLY
CANCEL

Show built-in Field List (Popup) over specific target

By passing the target element to the built-in field list dialog module in the [dataBound](#) event, the field list dialog will be displayed over the appropriate target element on a web page. By default, the Pivot Table's parent element is used as the target element to display the built-in field list dialog.

The sample code below shows the built-in field list dialog using `document.body` as the target element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from '../datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
})
```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
  width=width (dataBound)='ondataBound()' showFieldList='true'></ejs-
pivotview>`
  })
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  @ViewChild('pivotview',{static: false})
  public pivotGridObj?: PivotView;
  ondataBound(): void {
    (this.pivotGridObj as
PivotView).pivotFieldListModule.dialogRenderer.fieldListDialog.target =
document.body;
  }
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show field list using toolbar

It can also be viewed in toolbar by setting [showFieldList](#) and [showToolbar](#) properties in pivot table to **true**. Also, include the item **FieldList** within the [toolbar](#) property in pivot table. When toolbar is enabled, field list icon will be automatically added into the toolbar and the icon won't appear on top left corner in the pivot table component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'

```

```

import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, ConditionalFormattingService, PivotView,
  FieldListService, ToolbarService, ToolbarItems } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from '../datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, ToolbarService ],
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true' width=width
showToolbar='true' [toolbar]='toolbarOptions'></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  public toolbarOptions?: ToolbarItems[];
  ngOnInit(): void {
    this.toolbarOptions = ['FieldList'] as ToolbarItems[];
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from '../app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Invoking dynamic Field List (Customized)

Also, you can display the field list dialog independently through other means. For example, you can invoke the field list dialog on an external button click. To do so, set `renderMode` property to `Popup` and since on button click, field list dialog will be invoked.

* Meanwhile, you can display the field list dialog over specific target element within a web page using **target** property. By default, the **target** value is "null", which by default refers the **document.body** element.

* To make field list interact with pivot table, you need to use the **updateView** and **update** methods for data source update in both field list and pivot table simultaneously.

The below sample code illustrates the field list dialog invoked on an external button click.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
    EnginePopulatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from '../datasource';
@Component({
imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    styleUrls: ['./app.component.css'],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Popup"
(enginePopulated)='afterPopulate($event)' target='#fieldlisttarget'
allowCalculatedField='true' (load)='onLoad()'
(dataBound)='ondataBound()'></ejs-pivotfieldlist> <ejs-pivotview #pivotview
id='PivotView' height='530' (enginePopulated)='afterEnginePopulate($event)'
[gridSettings]='gridSettings'></ejs-pivotview> <div class="col-md-2"><button
ej-button id='fieldlistbtn'>Open Field List</button></div><div
id='fieldlisttarget'></div>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    public button?: Button;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldlistObj?: PivotFieldListComponent;
    afterPopulate(arge: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.updateView(this.pivotGridObj);
        }
    }
}
```



```

    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.update(this.pivotGridObj);
        }
    }
    onLoad(): void {
        if (Browser.isDevice) {
            (this.fieldlistObj as PivotFieldListComponent).renderMode =
'Popup';
            (this.fieldlistObj as PivotFieldListComponent).target =
'.control-section';
            (document.getElementById('PivotFieldList') as
HTMLElement).removeAttribute('style');
            setStyleAttribute(document.getElementById('PivotFieldList') as
HTMLElement, {
                'height': 0,
                'float': 'left'
            });
        }
    }
    ondataBound(): void {
        if (Browser.isDevice) {
            prepend([document.getElementById('PivotFieldList') as
HTMLElement], document.getElementById('PivotView') as HTMLElement);
        }
    }
    ngOnInit(): void {
        this.gridSettings = {
            columnWidth: 140
        } as GridSettings;
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.button = new Button({ isPrimary: true });
        this.button.appendTo('#fieldlistbtn');
        this.button.element.onclick = (): void => {
            (this.fieldlistObj as
PivotFieldListComponent).dialogRenderer.fieldListDialog.show();
        };
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Set caption to fields which isn't bound to the report

One can set the caption to all fields from the data source even if it is not bound to the actual report. It can be achieved using the [enginePopulated](#) event. On doing so, caption of the respective field will be displayed in both grouping bar and field list.

In the sample, we have set caption to the fields **Year** and **Quarter** dynamically.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotViewComponent, EnginePopulatedEventArgs } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true' width=width
(enginePopulated)='afterPopulate($event)'></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotViewComponent;
  afterPopulate(arg: EnginePopulatedEventArgs): void {
    Object.keys((this.pivotGridObj as
PivotViewComponent).engineModule.fieldList).forEach((key, index) => {
      if (key === 'Quarter') {
        (this.pivotGridObj as
PivotViewComponent).engineModule.fieldList[key].caption = 'Production
Quarter Year';
      }
      else if (key === 'Year') {
        (this.pivotGridObj as
PivotViewComponent).engineModule.fieldList[key].caption = 'Production Year';
      }
    });
  }
  ngOnInit(): void {
```

```

        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            columns: [{ name: 'Products' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show values button

During runtime, the **Values** button in the field list can be moved to a different position (i.e., different index) among other fields in the column or row axis. To enable the **Values** button, set the [showValuesButton](#) property to **true**.

This support is only available for relational data sources.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
    EnginePopulatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    styleUrls: ['./app.component.css'],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Fixed"
showValuesButton='true' (enginePopulated)='afterPopulate($event)'`

```

```

allowCalculatedField='true' (load)='onLoad()'
(dataBound)='ondataBound()'></ejs-pivotfieldlist>
<ejs-pivotview #pivotview id='PivotViewFieldList' width='99%' height='530'
(enginePopulated)='afterEnginePopulate($event)'
[gridSettings]='gridSettings'></ejs-pivotview>
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldlistObj?: PivotFieldListComponent;
    afterPopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.updateView(this.pivotGridObj);
        }
    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.update(this.pivotGridObj);
        }
    }
    onLoad(): void {
        if (Browser.isDevice) {
            (this.fieldlistObj as PivotFieldListComponent).renderMode =
'Popup';
            (this.fieldlistObj as PivotFieldListComponent).target =
'.control-section';
            (document.getElementById('PivotFieldList') as
HTMLElement).removeAttribute('style');
            setStyleAttribute(document.getElementById('PivotFieldList') as
HTMLElement, {
                'height': 0,
                'float': 'left'
            });
        }
    }
    ondataBound(): void {
        if (Browser.isDevice) {
            prepend([document.getElementById('PivotFieldList') as
HTMLElement], document.getElementById('PivotView') as HTMLElement);
        }
    }
    ngOnInit(): void {
        this.gridSettings = {
            columnWidth: 140
        } as GridSettings;
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],

```

```

        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Events

EnginePopulated

The [enginePopulated](#) event is available in both Pivot Table and Field List.

- The event [enginePopulated](#) is triggered in field list whenever the report gets modified. The updated report is passed to the pivot table via [updateView](#) method written within this event to refresh the same.
- Likewise, [enginePopulated](#) event is triggered in pivot table whenever the report gets modified. The updated report is passed to the field list via [update](#) method written within this event to refresh the same.

The event [enginePopulated](#) is triggered after engine is populated. It has following parameters - [dataSourceSettings](#), [pivotFieldList](#) and [pivotValues](#).

Note: This event is not required for Popup field list since it is a in built one.

FieldListRefreshed

The event [fieldListRefreshed](#) is triggered whenever there is any change done in the field list UI. It has following parameter - [dataSourceSettings](#) and [pivotValues](#). It allows user to identify each field list update. This event is applicable only for static field list.

For example, if we perform a sort operation within the field list, the field list will be refreshed. The [fieldListRefreshed](#) event will be triggered at that time and the user can perform custom operation inside that event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotViewComponent, FieldListRefreshedEventArgs } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,

```

```

        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    // specifies the template string for the pivot table component
    template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
(fieldListRefreshed)='fieldListRefreshed($event)' showFieldList='true'
width=width></ejs-pivotview></div>`
  })
  export class AppComponent {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    fieldListRefreshed(args: FieldListRefreshedEventArgs): void {
      //Triggers, whenever field list get refreshed.
    }
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

OnFieldDropped

The event [onFieldDropped](#) fires whenever a field is dropped in an axis. It has following parameters - `droppedAxis`, `droppedField` and `dataSourceSettings`. In this illustration, we have modified the `droppedField` caption through this event at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotViewComponent, FieldDroppedEventArgs } from '@syncfusion/ej2-angular-
pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
// specifies the template string for the pivot table component
template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
(onFieldDropped)='fieldDropped($event)' showFieldList='true'
width=width></ejs-pivotview></div>`
})
export class AppComponent {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    fieldDropped(args: FieldDroppedEventArgs | any): void {
        //Triggers, whenever field list get refreshed.
        args.droppedField.caption = args.droppedField.name + " --> " +
args.droppedAxis;
    }
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            allowLabelFilter: true,
            allowValueFilter: true,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ActionBegin

The event [actionBegin](#) triggers when the UI actions such as sorting, filtering, aggregation or edit calculated field, that are present in the field list UI begin. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. The following are the UI actions and their names:

Action	Action Name
-----	-----
Sort icon	Sort field
Filter icon	Filter field
Aggregation (Value type drop down and menu)	Aggregate field
Edit icon	Edit calculated field
Calculated field button	Open calculated field dialog
Field list	Open field list
Field list tree – Sort icon	Sort field tree

- **fieldInfo**: It holds the selected field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

- **cancel**: It allows user to restrict the current action.

In the below sample, opening pop-up field list can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotActionBeginEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [
```



```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    // specifies the template string for the pivot table component
    template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
(actionBegin)='actionBegin($event)' width=width></ejs-pivotview></div>`
  })
  export class AppComponent {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    actionBegin(args: PivotActionBeginEventArgs): void {
      if (args.actionName == 'Open field list') {
        args.cancel = true;
      }
    }
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionComplete

The event [actionComplete](#) triggers when the UI actions such as sorting, filtering, aggregation or edit calculated field, that are present in the field list UI, is completed. This allows user to identify the current UI actions being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.

- **actionName**: It holds the name of the current action completed. The following are the UI actions and their names:

| Action | Action Name |

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

| [Sort icon](#) | Field sorted |

| [Filter icon](#) | Field filtered |

| [Aggregation](#) (Value type drop down and menu) | Field aggregated |

| [Edit icon](#) | Calculated field edited |

| [Calculated field button](#) | Calculated field applied |

| [Field list](#) | Field list closed |

| [Field list tree – Sort icon](#) | Field tree sorted |

- **fieldInfo**: It holds the selected field information.

Note: This option is applicable only when the field based UI actions are performed such as filtering, sorting, removing field from grouping bar, editing and aggregation type change.

- **actionInfo**: It holds the unique information about the current UI action. For example, if sorting is completed, the event argument contains information such as sort order and the field name.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotActionCompleteEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
(actionComplete)= 'actionComplete($event)' width=width></ejs-
pivotview></div>`
})
export class AppComponent {
  public width?: string;
```

```

public dataSourceSettings?: IDataOptions;
actionComplete(args: PivotActionCompleteEventArgs): void {
    if (args.actionName == 'Field list closed') {
        // Triggers when the field list dialog is closed.
    }
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. The following are the UI actions and their names:

| Action | Action Name |

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

| [Sort icon](#) | Sort field |

| [Filter icon](#) | Filter field |

| [Aggregation](#) (Value type drop down and menu) | Aggregate field |

| [Edit icon](#) | Edit calculated field |

| [Calculated field button](#) | Open calculated field dialog |

| [Field list](#) | Open field list |

| [Field list tree – Sort icon](#) | Sort field tree |

- **errorInfo**: It holds the error information of the current UI action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
PivotActionFailureEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
// specifies the template string for the pivot table component
template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings showFieldList='true'
(actionFailure)='actionFailure($event)' width=width></ejs-pivotview></div>`
})
export class AppComponent {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    actionFailure(args: PivotActionFailureEventArgs): void {
        if (args.actionName == 'Open field list') {
            // Triggers when the current UI action fails to achieve the
desired result.
        }
    }
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            allowLabelFilter: true,
            allowValueFilter: true,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

See Also

- [Change load limited data in member editor](#)
- [Customize the icons for pivot table](#)

Defer update in Angular Pivotview component

Defer layout update support allows to update the pivot table component only on demand. On enabling this feature, end user can drag-and-drop fields between row, column, value and filter axes, apply sorting and filtering inside the Field List, resulting in change of pivot report alone but not the pivot table values. Once all operations are performed and on clicking the "Apply" button in the Field List, pivot table will start to update with the last modified report. This also helps in bringing better performance in pivot table component rendering.

The field list can be displayed in two different formats to interact with pivot table. They are:

- **In-built Field List (Popup):** To display the field list icon in pivot table UI to invoke the built-in dialog.
- **Stand-alone Field List (Fixed):** To display the field list in a static position within a web page.

In-built Field List (Popup)

To enable deferred updates in the pivot table, set the property [allowDeferLayoutUpdate](#) in pivot table as **true**. To make a note, the defer update option can be controlled only via Field List during runtime.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
```

```

    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true' width=width allowDeferLayoutUpdate='true'></ejs-
pivotview></div>`
  })
  export class AppComponent {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        allowLabelFilter: true,
        allowValueFilter: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Stand-alone Field List (Fixed)

The field list can be rendered in a static position, anywhere in web page layout, like a separate component. To do so, you need to set [renderMode](#) property to **Fixed** in pivot fieldlist.

To enable deferred updates in the static fieldlist, set the property [allowDeferLayoutUpdate](#) in fieldlist as **true**. To make a note, the defer update option can be controlled only via Field List during runtime.

To make field list interact with pivot table, you need to use the **UpdateView** and **Update** methods for data source update in both field list and pivot table simultaneously.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
EnginePopulatedEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';

```

```

import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    styleUrls: ['./app.component.css'],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Fixed"
(enginePopulated)='afterPopulate($event)' allowDeferLayoutUpdate='true'
allowCalculatedField='true' (load)='onLoad()'
(dataBound)='ondataBound()'></ejs-pivotfieldlist><ejs-pivotview #pivotview
id='PivotViewFieldList' allowDeferLayoutUpdate='true' width='99%'
height='530' (enginePopulated)='afterEnginePopulate($event)'
[gridSettings]='gridSettings'></ejs-pivotview>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldlistObj?: PivotFieldListComponent;
    afterPopulate(args: EnginePopulatedEventArgs): void {
        if ((this.fieldlistObj as PivotFieldListComponent).isRequiredUpdate)
        {
            (this.fieldlistObj as
PivotFieldListComponent).updateView(this.pivotGridObj as PivotViewComponent
);
        }
        this.pivotGridObj?.notify('ui-update', this.pivotGridObj);
        (this.fieldlistObj as PivotFieldListComponent).notify('tree-view-
update', this.fieldlistObj as PivotFieldListComponent);
    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldlistObj && this.pivotGridObj) {
            this.fieldlistObj.update(this.pivotGridObj);
        }
    }
    onLoad(): void {
        if (Browser.isDevice) {
            (this.fieldlistObj as PivotFieldListComponent).renderMode =
'Popup';
            (this.fieldlistObj as PivotFieldListComponent).target =
'.control-section';
            (document.getElementById('PivotFieldList') as
HTMLElement).removeAttribute('style');
            setStyleAttribute(document.getElementById('PivotFieldList') as
HTMLElement, {
                'height': 0,

```

```

        'float': 'left'
    });
    }
}
ondatabound(): void {
    if (Browser.isDevice) {
        prepend([document.getElementById('PivotFieldList') as
HTMLInputElement], document.getElementById('PivotView') as HTMLInputElement);
    }
}
ngOnInit(): void {
    this.gridSettings = {
        columnWidth: 140
    } as GridSettings;
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Performance

<!-- markdownlint-disable MD036 -->

Virtual scrolling in Angular Pivot Table component

Virtual Scrolling

The virtual scrolling option allows you to load the large amounts of data without performance degradation by rendering rows and columns only in the content viewport. The data will refresh dynamically on vertical or horizontal scroll. This feature can be enabled by setting the [enableVirtualization](#) property to **true**.

To use the virtual scrolling feature, inject the **VirtualScroll** module in to the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';

```



```

import { IDataOptions, IDataset, PivotView, VirtualScrollService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings enableVirtualization='true'
height='350'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public date1?: number;
  public date2?: number;
  data(count: number) {
    let result: Object[] = [];
    let dt: number = 0;
    for (let i: number = 1; i < count + 1; i++) {
      dt++;
      let round: string;
      let toString: string = i.toString();
      if (toString.length === 1) {
        round = "0000" + i;
      } else if (toString.length === 2) {
        round = "000" + i;
      } else if (toString.length === 3) {
        round = "00" + i;
      } else if (toString.length === 4) {
        round = "0" + i;
      } else {
        round = toString;
      }
      result.push({
        ProductID: "PRO-" + round,
        Year: "FY " + (dt + 2013),
        Price: Math.round(Math.random() * 5000) + 5000,
        Sold: Math.round(Math.random() * 80) + 10
      });
      if (dt / 4 === 1) {
        dt = 0;
      }
    }
    return result;
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: this.data(1000) as IDataset[],
      enableSorting: false,
      expandAll: true,
      formatSettings: [{ name: 'Price', format: 'C0' }],

```

```

        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Virtual scrolling with single page mode

When virtual scrolling is enabled, the pivot table renders not only the current view page, but also the previous and next pages by default. This default behavior, however, can cause performance delays when dealing with a large number of rows and columns. This is because the same number of rows and columns from adjacent pages are also processed, resulting in additional computational load. This performance constraint can be avoided by setting the [allowSinglePage](#) property to **true** within the [virtualScrollSettings](#).

Enabling this property causes the pivot table to render only the rows and columns that are relevant to the current view page during virtual scrolling. This optimization significantly improves the performance of the pivot table during initial rendering and when performing UI actions such as drill up/down, sorting, filtering, and more.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, VirtualScrollService,
VirtualScrollSettingsModel } from '@syncfusion/ej2-angular-pivotview';
@Component({
  imports: [

    PivotViewAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService],
  template: `<ejs-pivotview #pivotview id='PivotView' width='100%'
height='350'
[dataSourceSettings]='dataSourceSettings' enableVirtualization='true'
[virtualScrollSettings]='virtualScrollSettings'>
</ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public virtualScrollSettings?: VirtualScrollSettingsModel;
  data(count: number) {

```

```

let result: Object[] = [];
let dt: number = 0;
for (let i: number = 1; i < count + 1; i++) {
    dt++;
    let round: string;
    let toString: string = i.toString();
    if (toString.length === 1) {
        round = "0000" + i;
    } else if (toString.length === 2) {
        round = "000" + i;
    } else if (toString.length === 3) {
        round = "00" + i;
    } else if (toString.length === 4) {
        round = "0" + i;
    } else {
        round = toString;
    }
    result.push({
        ProductID: "PRO-" + round,
        Year: "FY " + (dt + 2013),
        Sold: Math.round(Math.random() * 80) + 10
    });
    if (dt / 4 == 1) {
        dt = 0;
    }
}
return result;
}
ngOnInit(): void {
    this.virtualScrollSettings = {
        allowSinglePage: true
    } as VirtualScrollSettingsModel;
    this.dataSourceSettings = {
        dataSource: this.data(1000) as IDataset[],
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' },
            { name: 'Sold', caption: 'Unit Sold' }
        ]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Limitations for virtual scrolling

- In virtual scrolling, the [columnWidth](#) property in [gridSettings](#) should be in pixels, and percentage values are not accepted.
- Resizing columns or setting width to individual columns affects the calculation used to pick the correct page on scrolling.
- Grouping, which takes additional time to splitting the raw items into the provided format.
- Date Formatting, which takes additional time to convert date format.
- Date Formatting with sorting, here additionally full date time format should be framed to perform sorting along with the provided date format which lags the performance.
- When using OLAP data, subtotals and grand totals are only displayed when measures are bound at the last position in the [rows](#) or [columns](#) axis. Otherwise, the data from the pivot table will be shown without summary totals.
- Even if virtual scrolling is enabled, not only is the current view port data retrieved, but also the data for the immediate previous page and the immediate next page. As a result, when the end user scrolls slightly ahead or behind, the next or previous page data is displayed immediately without requiring a refresh. **Note:** If the pivot table's width and height are large, the loading data count in the current, previous, and next viewports (pages) will also increase, affecting performance.

Virtual scrolling for static field list

Virtual scrolling automatically works with "Popup" field list on setting the [enableVirtualization](#) property in the Pivot Table to **true**. In case of static field list, which act as a separate component, user need to enable [enableVirtualization](#) property in the Pivot Table and also pass the report information to pivot table instance via the [load](#) event of the field list.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { PivotFieldListComponent, PivotViewComponent, FieldListService,
IDataOptions, IDataset,
    EnginePopulatedEventArgs, VirtualScrollService, PivotView } from
'@syncfusion/ej2-angular-pivotview';
import { Browser, setStyleAttribute, prepend } from '@syncfusion/ej2-base';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, VirtualScrollService],
  styleUrls: ['./app.component.css'],
  template: `<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings renderMode="Fixed"
(enginePopulated)= 'afterPopulate($event)' allowCalculatedField='true'
(load)= 'onLoad()' (dataBound)= 'ondataBound()'></ejs-pivotfieldlist>
    <ejs-pivotview #pivotview id='PivotViewFieldList' width='99%' height='530'
enableVirtualization='true'
' (enginePopulated)= 'afterEnginePopulate($event)'></ejs-pivotview>`
```

```

}))
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    data(count: number) {
        let result: Object[] = [];
        let dt: number = 0;
        for (let i: number = 1; i < count + 1; i++) {
            dt++;
            let round: string;
            let toString: string = i.toString();
            if (toString.length === 1) {
                round = "0000" + i;
            } else if (toString.length === 2) {
                round = "000" + i;
            } else if (toString.length === 3) {
                round = "00" + i;
            } else if (toString.length === 4) {
                round = "0" + i;
            } else {
                round = toString;
            }
            result.push({
                ProductID: "PRO-" + round,
                Year: "FY " + (dt + 2013),
                Price: Math.round(Math.random() * 5000) + 5000,
                Sold: Math.round(Math.random() * 80) + 10
            });
            if (dt / 4 == 1) {
                dt = 0;
            }
        }
        return result;
    }
    @ViewChild('pivotview', {static: false})
    public pivotObj?: PivotViewComponent;
    @ViewChild('pivotfieldlist')
    public fieldListObj?: PivotFieldListComponent;
    afterPopulate(arge: EnginePopulatedEventArgs): void {
        if (this.fieldListObj && this.pivotObj) {
            this.fieldListObj.updateView(this.pivotObj);
        }
    }
    afterEnginePopulate(args: EnginePopulatedEventArgs): void {
        if (this.fieldListObj && this.pivotObj) {
            this.fieldListObj.update(this.pivotObj);
        }
    }
    onLoad(): void {
        if (Browser.isDevice) {
            (this.fieldListObj as PivotFieldListComponent).renderMode = 'Popup';
            (this.fieldListObj as PivotFieldListComponent).target = '.control-
section';
            (document.getElementById('PivotFieldList') as
HTMLElement).removeAttribute('style');
            setStyleAttribute(document.getElementById('PivotFieldList') as
HTMLElement, {
                'height': 0,

```

```

        'float': 'left'
    });
    }
    (this.fieldListObj as PivotFieldListComponent).pivotGridModule =
this.pivotObj as PivotViewComponent;
    //Assigning report to pivot table component
    (this.pivotObj as PivotViewComponent).dataSourceSettings =
(this.fieldListObj as PivotFieldListComponent).dataSourceSettings;
    //Generating page settings based on pivot table component's size.
    (this.pivotObj as PivotViewComponent).updatePageSettings(true);
    //Assigning page settings to field list component.
    (this.fieldListObj as PivotFieldListComponent).pageSettings =
(this.pivotObj as PivotViewComponent).pageSettings;
    }
    ondataBound(): void {
        if (Browser.isDevice) {
            prepend([document.getElementById('PivotFieldList') as HTMLElement],
document.getElementById('PivotView') as HTMLElement);
        }
    }
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: this.data(1000) as IDataset[],
            enableSorting: false,
            expandAll: true,
            formatSettings: [{ name: 'Price', format: 'C0' }],
            rows: [{ name: 'ProductID' }],
            columns: [{ name: 'Year' }],
            values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See also

- [Paging in Syncfusion EJ2 Typescript components](#)
- [Data Compression in Syncfusion EJ2 Typescript components](#)

Paging in Angular Pivotview component

Paging allows you to load large amounts of data that can be divided and displayed page by page in the pivot table. It can be enabled by setting the [enablePaging](#) property to **true**. It can be configured at code-behind by using the [pageSettings](#) property, during initial rendering of the component. The properties required are:

- [currentRowPage](#): Allows user to set the current row page number to be displayed in the pivot table.
- [currentColumnPage](#): Allows user to set the current column page number to be displayed in the pivot table.
- [rowPageSize](#): Allows user to set the total number of records to be displayed on each page of the pivot table's row axis.
- [columnPageSize](#): Allows user to set the total number of records to be displayed on each page of the pivot table's column axis.

Pager UI

When paging is enabled, a built-in pager UI appears at the bottom of the pivot table, allowing you to change the current page in the row and column axes by navigating to a desired page using the navigation buttons or an input text box, as well as change the page size via dropdown at runtime.

You can also change the position, visibility, compact view, and template of the row and column pagers by using the [pagerSettings](#).

In order to see and use the pager UI, insert the `PagerService` module into the pivot table using the `@NgModule.providers` section.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
  providers: [PagerService]
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public remoteData?: DataManager;
  public width?: string;
  public gridSettings?: GridSettings;
  public pageSettings?: PageSettings;
  public pagerSettings?: PagerSettings;
```

```

ngOnInit(): void {
    this.remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
        rowPageSize: 10,
        columnPageSize: 5,
        currentColumnPage: 1,
        currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
        position: 'Bottom',
        enableCompactView: false,
        showColumnPager: true,
        showRowPager: true,
        columnPageSizes: [5, 10, 20, 50, 100],
        rowPageSizes: [10, 50, 100, 200],
        isInversed: false,
        showColumnPageSize: true,
        showRowPageSize: true
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
        dataSource: this.remoteData,
        expandAll: true,
        filters: [],
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show pager UI at top or bottom

You can display the pager UI at top or bottom of the pivot table by using the [position](#) property. To show the pager UI at top of the pivot table, set the [position](#) property in [pagerSettings](#) to **Top**.

By default, the pager UI appears at the bottom of the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```



```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
providers: [PagerService]
})
export class AppComponent implements OnInit {
public dataSourceSettings?: IDataOptions;
public remoteData?: DataManager;
public width?: string;
public gridSettings?: GridSettings;
public pageSettings?: PageSettings;
public pagerSettings?: PagerSettings;
ngOnInit(): void {
    this.remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
        rowPageSize: 10,
        columnPageSize: 5,
        currentColumnPage: 1,
        currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
        position: 'Top'
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
        dataSource: this.remoteData,
        expandAll: true,
        filters: [],
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],

```

```

        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Inverse pager

Toggles and displays row and column pager. To show the column pager on the left side of the pager UI, set the `isInversed` property in `pagerSettings` to `true`.

By default, the row pager is displayed on the left side of the pager UI, while the column pager is displayed on the right side.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
providers: [PagerService]
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public remoteData?: DataManager;
    public width?: string;
    public gridSettings?: GridSettings;
    public pageSettings?: PageSettings;
    public pagerSettings?: PagerSettings;
    ngOnInit(): void {

```

```

    this.remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
        rowPageSize: 10,
        columnPageSize: 5,
        currentColumnPage: 1,
        currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
        isInversed: true
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
        dataSource: this.remoteData,
        expandAll: true,
        filters: [],
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Compact view

By hiding all except the previous and next navigation buttons, the pager UI can be displayed with the absolute minimum of paging options. The compact view can be enabled by setting the [enableCompactView](#) property in [pagerSettings](#) to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';

```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
  providers: [PagerService]
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public remoteData?: DataManager;
  public width?: string;
  public gridSettings?: GridSettings;
  public pageSettings?: PageSettings;
  public pagerSettings?: PagerSettings;
  ngOnInit(): void {
    this.remoteData = new DataManager({
      url: 'https://bi.syncfusion.com/northwindservice/api/orders',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
      rowPageSize: 10,
      columnPageSize: 5,
      currentColumnPage: 1,
      currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
      enableCompactView: true
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
      dataSource: this.remoteData,
      expandAll: true,
      filters: [],
      columns: [{ name: 'ProductName', caption: 'Product Name' }],
      rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
      formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
      values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Show or hide paging option

By using the [showRowPager](#) and [showColumnPager](#) properties in [pagerSettings](#), you can show or hide row and column pager separately in the pager UI.

In the following example, row pager has been disabled by setting the [showRowPager](#) property in [pagerSettings](#) to **false**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
providers: [PagerService]
})
export class AppComponent implements OnInit {
public dataSourceSettings?: IDataOptions;
public remoteData?: DataManager;
public width?: string;
public gridSettings?: GridSettings;
public pageSettings?: PageSettings;
public pagerSettings?: PagerSettings;
ngOnInit(): void {
    this.remoteData = new DataManager({
        url: 'https://bi.syncfusion.com/northwindservice/api/orders',
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
        rowPageSize: 10,
        columnPageSize: 5,
```

```

        currentColumnPage: 1,
        currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
        showRowPager: false
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
        dataSource: this.remoteData,
        expandAll: true,
        filters: [],
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show or hide page size

By using the [showRowPageSize](#) and [showColumnPageSize](#) properties in [pagerSettings](#), you can show or hide "Rows per page" and "Columns per page" dropdown menu. The dropdown menu contains a list of pre-defined or user-defined page sizes, which will be displayed in the "Rows per page" and "Columns per page" dropdowns, allowing you to change the page size for the row and column axes at runtime.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',

```

```

    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
    providers: [PagerService]
  })
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public remoteData?: DataManager;
  public width?: string;
  public gridSettings?: GridSettings;
  public pageSettings?: PageSettings;
  public pagerSettings?: PagerSettings;
  ngOnInit(): void {
    this.remoteData = new DataManager({
      url: 'https://bi.syncfusion.com/northwindservice/api/orders',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
      rowPageSize: 10,
      columnPageSize: 5,
      currentColumnPage: 1,
      currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
      showColumnPageSize: false,
      showRowPageSize: false
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
      dataSource: this.remoteData,
      expandAll: true,
      filters: [],
      columns: [{ name: 'ProductName', caption: 'Product Name' }],
      rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
      formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
      values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize page size

By using the [rowPageSizes](#) and [columnPageSizes](#) properties in [pagerSettings](#), you can specify a set of desired page sizes, which will be displayed in the "Rows per page" and "Columns per page" dropdowns, allowing you to change the page size for the row and column axes at runtime.

By default, the "Rows per page" dropdown have pre-defined page sizes of **10, 50, 100, and 200**, while the "Columns per page" dropdown have pre-defined page sizes of **5, 10, 20, 50, and 100**.

In the following example, the "Rows per page" dropdown is set with user-defined page sizes of **10, 20, 30, 40, and 50** and the "Columns per page" dropdown is set with user-defined page sizes of **5, 10, 15, 20, and 30**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PagerSettings, PageSettings,
PagerService } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
'@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"></ejs-
pivotview>`,
  providers: [PagerService]
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public remoteData?: DataManager;
  public width?: string;
  public gridSettings?: GridSettings;
  public pageSettings?: PageSettings;
  public pagerSettings?: PagerSettings;
  ngOnInit(): void {
    this.remoteData = new DataManager({
      url: 'https://bi.syncfusion.com/northwindservice/api/orders',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.width = '100%';
    this.pageSettings = {
      rowPageSize: 10,
      columnPageSize: 5,
```



```

        currentColumnPage: 1,
        currentRowPage: 1
    } as PageSettings;
    this.pagerSettings = {
        position: 'Bottom',
        enableCompactView: false,
        showColumnPager: true,
        showRowPager: true,
        columnPageSizes: [5, 10, 15, 20, 30],
        rowPageSizes: [10, 20, 30, 40, 50],
        isInversed: false,
        showColumnPageSize: true,
        showRowPageSize: true
    } as PagerSettings;
    this.gridSettings = { columnWidth: 120 } as GridSettings;
    this.dataSourceSettings = {
        dataSource: this.remoteData,
        expandAll: true,
        filters: [],
        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Template

The [template](#) property allows to change the appearance of the pager UI by displaying user-defined HTML elements instead of built-in HTML elements.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, PagerSettings, PageSettings, PagerService
} from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { Pager } from '@syncfusion/ej2-grids';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({

```

```

imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]='gridSettings' enablePaging="true"
[pageSettings]="pageSettings" [pagerSettings]="pagerSettings"
(dataBound)='dataBound()' '></ejs-pivotview>`,
providers: [PagerService]
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public remoteData?: DataManager;
    public width?: string;
    public gridSettings?: GridSettings;
    public pageSettings?: PageSettings;
    public pagerSettings?: PagerSettings;
    public rowPager?: Pager;
    public columnPager?: Pager;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    dataBound() {
        this.updateTemplate();
    }
    updateTemplate() {
        if (!isNullOrUndefined(this.rowPager as Pager)) {
            this.rowPager?.destroy();
            this.rowPager = undefined;
        }
        this.rowPager = new Pager({
            pageSize: this.pivotGridObj?.pageSettings.rowPageSize,
            totalRecordsCount: this.pivotGridObj?.engineModule.rowCount,
            currentPage: this.pivotGridObj?.pageSettings.currentRowPage,
            pageCount: 5,
            click: this.rowPageClick.bind(this)
        });
        this.rowPager.appendTo('#row-pager');
        if (!isNullOrUndefined(this.columnPager as Pager)) {
            this.columnPager?.destroy();
            this.columnPager = undefined;
        }
        this.columnPager = new Pager({
            pageSize: this.pivotGridObj?.pageSettings.columnPageSize,
            totalRecordsCount: this.pivotGridObj?.engineModule.columnCount,
            currentPage: this.pivotGridObj?.pageSettings.currentColumnPage,
            pageCount: 5,
            click: this.columnPageClick.bind(this)
        });
        this.columnPager.appendTo('#column-pager');
    }
    rowPageClick(args: any) {
        this.pivotGridObj!.pageSettings.currentRowPage = args.currentPage;
        this.pivotGridObj?.refreshData();
    }
}

```

```

    }
    columnPageClick(args: any) {
        this.pivotGridObj!.pageSettings.currentColumnPage =
args.currentPage;
        this.pivotGridObj?.refreshData();
    }
    ngOnInit(): void {
        this.remoteData = new DataManager({
            url: 'https://bi.syncfusion.com/northwindservice/api/orders',
            adaptor: new WebApiAdaptor,
            crossDomain: true
        });
        this.width = '100%';
        this.pageSettings = {
            rowPageSize: 10,
            columnPageSize: 5,
            currentColumnPage: 1,
            currentRowPage: 1
        } as PageSettings;
        this.pagerSettings = {
            template: '#template'
        } as PagerSettings;
        this.gridSettings = { columnWidth: 120 } as GridSettings;
        this.dataSourceSettings = {
            dataSource: this.remoteData,
            expandAll: true,
            filters: [],
            columns: [{ name: 'ProductName', caption: 'Product Name' }],
            rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
            formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
            values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Data Compression in Angular Pivot Table component

This property is applicable only for the relational data source.

When binding one million raw data, the pivot table processes all raw data to generate aggregated data during initial rendering and report manipulation. However, with data compression, the input raw data is compressed based on the uniqueness of the raw data, and the final compressed raw data are utilized by the pivot table. The compressed raw data is then used for further operations at all times, reducing the looping complexity and improving the performance of the pivot table. For example, if the pivot table is connected to one million raw data compressed to 1,000 unique raw data, it will render within 3 seconds

rather than 10 seconds. You can enable this option by using the [allowDataCompression](#) property along with the [enableVirtualization](#) property.

This options will only function when the virtual scrolling is enabled.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings allowDataCompression='true'
enableVirtualization='true' height='350'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public date1?: number;
  public date2?: number;
  data(count: number) {
    let result: Object[] = [];
    let dt: number = 0;
    for (let i: number = 1; i < count + 1; i++) {
      dt++;
      let round: string;
      let toString: string = i.toString();
      if (toString.length === 1) {
        round = "0000" + i;
      } else if (toString.length === 2) {
        round = "000" + i;
      } else if (toString.length === 3) {
        round = "00" + i;
      } else if (toString.length === 4) {
        round = "0" + i;
      } else {
        round = toString;
      }
      result.push({
        ProductID: 'PRO-' + (i % 1000),
        Year: "FY " + (dt + 2013),
        Price: Math.round(Math.random() * 5000) + 5000,
        Sold: Math.round(Math.random() * 80) + 10
      });
    }
  }
}
```

```

    if (dt / 4 == 1) {
        dt = 0;
    }
}
return result;
}

ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: this.data(1000000) as IDataset[],
        enableSorting: false,
        expandAll: true,
        formatSettings: [{ name: 'Price', format: 'C0' }],
        rows: [{ name: 'ProductID' }],
        columns: [{ name: 'Year' }],
        values: [{ name: 'Price', caption: 'Unit Price' }, { name: 'Sold',
caption: 'Unit Sold' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Limitations during data compression:

- The following aggregation types will not be supported:
- Average
- Populationstdev
- Samplestdev
- Populationvar
- Samplevar
- If you use any of the above aggregations, they will result in the aggregation type **“Sum”**.
- **“DistinctCount”** will act as **“Count”** aggregation type.
- In the calculated field, an existing field can be inserted without altering its default aggregation type. Even if changed, it would revert to the default aggregation type for calculation.

State persistence in Angular Pivotview component

State persistence allows user to maintain the current state of the component along with its report bounded in the browser local storage (cookie). Even if the browser is refreshed or if you move to the next page within the browser, components state will be persisted. State persistence stores the Pivot Table object in the local storage when [enablePersistence](#) property in pivot table is set to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'

```

```

import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotViewComponent } from '@syncfusion/ej2-
angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='300'
[dataSourceSettings]=dataSourceSettings enablePersistence='true' ></ejs-
pivotview></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotViewComponent;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Save and Load Pivot Layout

You can save the current layout of the pivot table by using [getPersistData](#) in string format. The saved layout can be loaded to pivot table any time by passing the saved data as a parameter to [Link to the Video](#) method in the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { DisplayOption, IDataOptions, IDataset, PivotViewComponent } from
'@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<span><button ej-button
id='save'>Save</button></span><span><button ej-button
id='load'>Load</button></span><div><ejs-pivotview #pivotview id='PivotView'
height='300' [dataSourceSettings]=dataSourceSettings showGroupingBar='true'
></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public chartSettings?: ChartSettings;
  public displayOption?: DisplayOption;
  public saveButton?: Button;
  public loadButton?: Button;
  public layout?: string;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotViewComponent;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.saveButton = new Button({ isPrimary: true });
    this.saveButton.appendTo('#save');
    this.saveButton.element.onclick = (): void => {
      this.layout = this.pivotGridObj?.getPersistData();
    };
    this.loadButton = new Button({ isPrimary: true });
    this.loadButton.appendTo('#load');
    this.loadButton.element.onclick = (): void => {
      this.pivotGridObj?.loadPersistData(this.layout as string);
    };
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

<!-- markdownlint-disable MD012 -->

Row and column in Angular Pivotview component

To learn about how to use the row and column options effectively in the Angular Pivot Table, watch this video:

Width and Height

Allows end user to set the pivot table's height and width by using [height](#) and [width](#) properties in pivot table respectively. The supported formats to set [height](#) and [width](#) properties are,

- Pixel: For example - 100, 200, "100px", "200px".
- Percentage: For example - "100%", "200%".
- Auto: It is applicable for [height](#) property alone in-order to render the pivot table beyond its parent container height without vertical scrollbar. The parent container here would show its vertical scrollbar as soon as the component reaches beyond its dimension.

The pivot table will not be displayed less than **400px**, since it's the minimum width of the component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
```



```

        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Row Height

Allows end user to set the height of each pivot table rows commonly using the [rowHeight](#) property in [gridSettings](#).

By default, the [rowHeight](#) property is set as **36** pixels for desktop layout and **48** pixels for mobile layout.

The height of the column headers alone may vary when grouping bar feature is enabled.

In the below code sample, the [rowHeight](#) property is set as **60** pixels.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;

```

```

public dataSourceSettings?: IDataOptions;
public gridSettings?: GridSettings;
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.gridSettings = {
        rowHeight: 60
    } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Column Width

Allows end user to set the width of each pivot table columns commonly using the [columnWidth](#) property in [gridSettings](#).

By default, the [columnWidth](#) property is set as **110** pixels to each columns except the first column. For first column, **250** pixels and **200** pixels are set respectively with and without grouping bar.

In the below example, the [columnWidth](#) property is set as **200** pixels.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',

```

```
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
))
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.gridSettings = {
            columnWidth: 120
        } as GridSettings;
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Adjust width based on columns

By default, if the component width set in code-behind is more than the width of the total columns, then the columns will be stretched to make it fit. To avoid the stretching, set the [allowAutoResizing](#) property in the [gridSettings](#) to **false**. By doing so, the component will be adjusted (shrunk) based on the width of total columns.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
    imports: [
```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }],
        rows: [{ name: 'Country' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filterSettings: [{ name: 'Year', type: 'Exclude', items: ['FY
2015', 'FY 2017'] }],
        filters: []
      };
      this.gridSettings = {
        allowAutoResizing: false
      } as GridSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Reorder

Allows end user to reorder a particular column header from one index to another index within the pivot table through drag-and-drop option. It can be enabled by setting the [allowReordering](#) property in [gridSettings](#) to true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';

```

```

import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataSet[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      allowReordering: true
    } as GridSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Column Resizing

Allows end user to resize the columns by clicking and dragging the right edge of the column header. While dragging, the width of the respective column will be resized immediately. To enable column resizing option, set the [allowResizing](#) property in [gridSettings](#) to **true**.

By default, the column resizing option is enabled.

In RTL mode, user can click and drag the left edge of the header cell to resize the column.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      allowResizing: true
    } as GridSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Text Wrap

Allows end user to wrap the cell content to the next line when it exceeds the boundary of the cell width. To enable text wrap, set the [allowTextWrap](#) property in [gridSettings](#) to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      allowTextWrap: true
    } as GridSettings;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Text Align

Allows end user to align the content of the pivot table's row and column headers and value cells by using the [textAlign](#) and [headerTextAlign](#) properties in the [columnRender](#) event under [gridSettings](#). The following alignments are:

- **Left** - It allows the content to be positioned on the left.
- **Right** - It allows the content to be positioned on the right.
- **Center** - It allows the content to be positioned in the middle.
- **Justify** - It allows the content to be as flexible as possible, when the cell does not occupy the entire available area.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
import { Observable } from 'rxjs';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public observable = new Observable();
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
```



```

        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.gridSettings = {
        columnWidth: 140,
        columnRender: this.observable.subscribe((args: any) => {
            if ((args as any).stackedColumns[0]) {
                // Content for the row headers is right-aligned here.
                (args as any).stackedColumns[0].textAlign = 'Right';
            }
            if ((args as any).stackedColumns[1]) {
                // Content for the column header "FY 2015" is center-
aligned here.
                (args as any).stackedColumns[1].textAlign = 'Center';
            }
            if ((args as any).stackedColumns[1] && ((args as
any).stackedColumns[1].columns[0] as any)) {
                // Content for the column header "Q1" is right-aligned
here.
                ((args as any).stackedColumns[1].columns[0] as
any).textAlign = 'Right';
            }
            if ((args as any).stackedColumns[1] && (args as
any).stackedColumns[1].columns[0] && ((args as
any).stackedColumns[1].columns[0] as any).columns[0]) {
                // Content for the value header "Units Sold" is right-
aligned here.
                ((args as any).stackedColumns[1].columns[0] as
any).columns[0].headerTextAlign = 'Right';
            }
            if ((args as any).stackedColumns[1] && (args as
any).stackedColumns[1].columns[0] && ((args as
any).stackedColumns[1].columns[0] as any).columns[0]) {
                // Content for the values are left-aligned here.
                ((args as any).stackedColumns[1].columns[0] as
any).columns[0].textAlign = 'Left';
            }
        }) as any
    } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

AutoFit

Allows the user to fit the Pivot Table columns as wide as the content of the cell without wrapping. It auto fits all of the Pivot Table columns by invoking the [autoFitColumns](#) method from the grid instance.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
width=width (dataBound)='ondataBound()'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ondataBound(): void {
    this.pivotGridObj?.grid.autoFitColumns();
  }
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The minimum width of 250 pixels is set by default with the grouping bar UI for the first column and cannot be reduced further. So, when the grouping bar is enabled, one can auto fit the Pivot Table columns by calling the [autoFitColumns](#) method from the grid instance with the parameter contained pivot table columns field name excluding first column.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
width=width (dataBound)='ondataBound()'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview',{static: false})
    public pivotGridObj?: PivotView;
    ondataBound(): void {
        if (this.pivotGridObj?.showGroupingBar) {
            let columns: string[] = [];
            for (let i: number = 1; i < (this.pivotGridObj?.grid as
any).columnModel.length; i++) {
                columns.push((this.pivotGridObj?.grid as
any).columnModel[i].field);
            }
            this.pivotGridObj?.grid.autoFitColumns(columns);
        }
    }
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],

```

```

        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Autofit specific columns

During initial rendering, the parameter `autoFit` in the `columnRender` event under `gridSettings` can be set to `true` to autofit specific columns.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Observable } from 'rxjs';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
width=width (columnRender)=columnRender($event)></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  columnRender($event: Event) {
  }
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public observable = new Observable();
  ngOnInit(): void {
    this.gridSettings = {
      columnRender: this.observable.subscribe((args: any) => {
        for(var i = 0; i < (args as any).columns.length;i++) {
          (args as any).columns[i].autoFit = true;
        }
      })
    }
  }
}

```

```

    }) as any
  } as GridSettings;
  this.width = '100%';
  this.dataSourceSettings = {
    dataSource: Pivot_Data as IDataset[],
    expandAll: true,
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Grid Lines

Allows end user to display cell border for each cells using [gridLines](#) property in [gridSettings](#).

Available mode of grid lines are:

| Modes | Actions |

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

| Both | Displays both the horizontal and vertical grid lines. |

| None | No grid lines are displayed. |

| Horizontal | Displays the horizontal grid lines only. |

| Vertical | Displays the vertical grid lines only. |

| Default | Displays grid lines based on the theme. |

By default, pivot table renders grid lines in **Both** mode.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.gridSettings = {
        gridLines: 'Vertical'
      } as GridSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Selection

Selection provides an option to highlight a row or a column or a cell. It can be done through simple mouse down or arrow keys. To enable selection in the pivot table, set the [allowSelection](#) property in [gridSettings](#) to **true**.

The pivot table supports two types of selection that can be set using [type](#) property in [selectionSettings](#). The selection types are:

- **Single:** It is set by default, and it only allows selection of a single row or a column or a cell.
- **Multiple:** Allows you to select multiple rows or columns or cells.

To perform multi-selection, press and hold "CTRL" key and click the desired rows or cells. To select range of rows or cells, press and hold the "SHIFT" key and click the rows or columns or cells.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      allowSelection: true,
      selectionSettings: { type: 'Multiple' }
    } as GridSettings;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Selection Mode

The pivot table supports four types of selection mode that can be set using [mode](#) in [selectionSettings](#). The selection modes are:

- **Row:** It is set by default, and allows user to select only rows.
- **Column:** Allows user to select only columns.
- **Cell:** Allows user to select only cells.
- **Both:** Allows user to select rows and columns at the same time.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
};
```



```

        this.gridSettings = {
            allowSelection: true,
            selectionSettings: { mode: 'Both' }
        } as GridSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Cell Selection Mode

The pivot table supports two types of cell selection mode that can be set using [cellSelectionMode](#) in [selectionSettings](#). The cell selection modes are:

- **Flow:** It is set by default. The range of cells are selected between the start index and end index that includes in-between cells of rows.
- **Box:** Range of cells are selected from the start and end column indexes that includes in-between cells of rows within the range.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview';
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {

```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.gridSettings = {
        allowSelection: true,
        selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple',
mode: 'Cell' }
    } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Cell selection requires [mode](#) property in [selectionSettings](#) to be **Cell** or **Both**, and [type](#) property should be **Multiple**.

Changing background color of the selected cell

The background-color of the selected cell can be changed using built-in CSS names. To do so, please refer to the code sample below, which shows that the selected cells are changed to a **green yellow** color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component

```

```

    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
    [gridSettings]='gridSettings' width=width></ejs-pivotview>`,
    styleUrls: ['./app.component.css']
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.gridSettings = {
        allowSelection: true,
        selectionSettings: { cellSelectionMode: 'Box', type: 'Multiple',
mode: 'Cell' }
      } as GridSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Event

CellSelected

The event `cellSelected` is triggered when cell selection gets completed. It provides selected cells information with its corresponding column and row headers. It has following parameters - `selectedCellsInfo`, `currentCell` and `target`. This event allows user to view selected cells information and user can pass those selected cells information to any external component for data binding.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotCellSelectedEventArgs } from
'@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';

```

```

import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]=gridSettings (cellSelected)='cellSelected($event)'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      formatSettings: [{ name: 'Amount', format: 'C2', useGrouping:
false,
        minimumSignificantDigits: 1, maximumSignificantDigits: 3
    }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      filters: []
    };
    this.gridSettings = {
      allowSelection: true,
      selectionSettings: { mode: 'Both', type: 'Multiple' }
    } as GridSettings;
  }
  cellSelected(args: PivotCellSelectedEventArgs){
    //args.selectedCellsInfo -> get selected cells information.
    //args.pivotValues -> get the pivot values of the pivot table.
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

CellSelecting

The event `cellSelecting` triggers before cell gets selected gets completed. It provides selected cells information with its corresponding column and row headers. It has following parameters - `currentCell`, `data` and `cancel`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotCellSelectedEventArgs } from
'@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[gridSettings]=gridSettings (cellSelecting)= 'cellSelecting($event)'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  public gridSettings?: GridSettings;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      formatSettings: [{ name: 'Amount', format: 'C2', useGrouping:
false,
        minimumSignificantDigits: 1, maximumSignificantDigits: 3
}],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      filters: []
    };
    this.gridSettings = {
      allowSelection: true,
      selectionSettings: { mode: 'Both', type: 'Multiple' }
    } as GridSettings;
```

```

    }
    cellSelecting(args: PivotCellSelectedEventArgs) {
        //args.selectedCellsInfo -> get selected cells information.
        //args.pivotValues -> get the pivot values of the pivot table.
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Clip Mode

The clip mode provides options to display its overflow cell content in the pivot table. It can be configured using the [clipMode](#) property in [gridSettings](#). The pivot table supports three types of clip modes which are:

- **Clip:** Truncates the cell content when it overflows its area.
- **Ellipsis:** Displays ellipsis when the cell content overflows its area.
- **EllipsisWithTooltip:** Displays ellipsis when the cell content overflows its area, also it will display the tooltip while hover on ellipsis is applied.

By default, [clipMode](#) value is set to **Ellipsis**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;

```

```

ngOnInit(): void {
  this.width = '100%';
  this.dataSourceSettings = {
    dataSource: Pivot_Data as IDataset[],
    expandAll: false,
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  this.gridSettings = {
    clipMode: 'Clip'
  } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Cell Template

User can customize the pivot table cell element by using the `cellTemplate` property in pivot table. The `cellTemplate` property accepts either an HTML string or the element's ID, which can be used to append additional HTML elements to showcase each cell with custom format.

In this demo, the revenue cost for each year is represented with trend icons.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview';
import { Component, ViewChild, OnInit } from '@angular/core';
import { IDataOptions, PivotView, IAxisSet, IFieldOptions,
PivotViewComponent, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { renewableEnergy } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component

```

```

    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings (dataBound)='trend()' width=width
height=height [cellTemplate]=cellTemplate></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public height?: number;
    public dataSourceSettings?: IDataOptions;
    public cellTemplate?: string;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotViewComponent;
    trend(): void {
      let cTable: HTMLElement[] =
[[]].slice.call(document.getElementsByClassName("e-table"));
      let colLen: number = this.pivotGridObj?.pivotValues[3].length as
number;
      let cLen: number = cTable[3].children[0].children.length;
      let rLen: number = cTable[3].children[1].children.length;
      let rowIndx: number;
      for (let k: number = 0; k < rLen; k++) {
        if (this.pivotGridObj?.pivotValues[k] &&
this.pivotGridObj?.pivotValues[k][0] !== undefined) {
          rowIndx = ((this.pivotGridObj?.pivotValues[k][0]) as
IAxisSet).RowIndex as number;
          break;
        }
      }
      let rowHeaders: HTMLElement[] =
[[]].slice.call(cTable[2].children[1].querySelectorAll('td'));
      let rows: IFieldOptions[] =
this.pivotGridObj?.dataSourceSettings.rows as IFieldOptions[];
      if (rowHeaders.length > 1) {
        for (let i: number = 0, Cnt = rows; i < Cnt.length; i++) {
          let fields: any = {};
          let fieldHeaders: any = [];
          for (let j: number = 0, Lnt = rowHeaders; j < Lnt.length;
j++) {
            let header: any = rowHeaders[j];
            if (header.className.indexOf('e-gtot') === -1 &&
header.className.indexOf('e-rowsheader') > -1 &&
header.getAttribute('fieldname') === rows[i].name) {
              var headerName =
rowHeaders[j].getAttribute('fieldname') + '_' + rowHeaders[j].textContent;
              fields[rowHeaders[j] as number].textContent as
string = j;
              fieldHeaders.push(rowHeaders[j].textContent);
            }
          }
          if (i === 0) {
            for (let rnt: number = 0, Lnt = fieldHeaders; rnt <
Lnt.length; rnt++) {
              if (rnt !== 0) {
                let row: number = fields[fieldHeaders[rnt]];
                let prevRow: number = fields[fieldHeaders[rnt -
1]];
                for (let j: number = 0, ci = 1; j < cLen && ci <
colLen; j++ , ci++) {

```



```

        let node: HTMLElement =
cTable[3].children[1].children[row].childNodes[j] as HTMLElement;
        let prevNode: HTMLElement =
cTable[3].children[1].children[prevRow].childNodes[j] as HTMLElement;
        let ri: any = undefined;
        let prevRi: any = undefined;
        if (node) {
            ri = node.getAttribute("index");
        }
        if (prevNode) {
            prevRi = prevNode.getAttribute("index");
        }
        if (ri && ri <
[0].slice.call(this.pivotGridObj?.pivotValues).length) {
            if
(((this.pivotGridObj?.pivotValues[prevRi][ci] as IAxisSet).value as number)
> ((this.pivotGridObj?.pivotValues[ri][ci] as IAxisSet).value as number) &&
node.querySelector('.tempwrap')) {
                let trendElement: HTMLElement =
node.querySelector('.tempwrap') as HTMLElement;
                trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-loss');
            } else if
(((this.pivotGridObj?.pivotValues[prevRi][ci] as IAxisSet).value as number)
< ((this.pivotGridObj?.pivotValues[ri][ci] as IAxisSet).value as number) &&
node.querySelector('.tempwrap')) {
                let trendElement: HTMLElement =
node.querySelector('.tempwrap') as HTMLElement;
                trendElement.className =
trendElement.className.replace('sb-icon-neutral', 'sb-icon-profit');
            }
        }
    }
}

ngOnInit(): void {
    this.width = "100%";
    this.height = 350;
    this.cellTemplate = '<span class="tempwrap sb-icon-neutral pv-
icons"></span>';
    this.dataSourceSettings = {
        dataSource: renewableEnergy as IDataSet[],
        expandAll: true,
        enableSorting: true,
        drilledMembers: [{ name: 'Year', items: ['FY 2015', 'FY 2017',
'FY 2018'] }],
        formatSettings: [{ name: 'ProCost', format: 'C0' }],
        rows: [
            { name: 'Year', caption: 'Production Year' }
        ],
        columns: [
            { name: 'EnerType', caption: 'Energy Type' },
            { name: 'EneSource', caption: 'Energy Source' }
        ]
    }
}

```

```

        ],
        values: [
            { name: 'ProCost', caption: 'Revenue Growth' }
        ],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Events

QueryCellInfo

The event `queryCellInfo` triggers while rendering row and value cells in the pivot table. It allows the user to customize the element of the current cell. It has the following parameters:

- `cell` - It holds the current cell info
- `data` - It holds entire row data
- `column` - It holds column information for the current cell
- `pivotview` - It holds pivot instance

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview';
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { Pivot_Data } from './datasource';
@Component({
    imports: [
        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
})

```

```

export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    public columnGrandTotalIndex?: number;
    public rowGrandTotalIndex?: number;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    queryCell(args: any): void {
        ((this.pivotGridObj as PivotView).renderModule as
any).rowCellBoundEvent(args);
        //triggers for every cell
    }
    enginePopulated(args: any): void {
        ((this.pivotGridObj as PivotView).grid as Grid).queryCellInfo =
this.queryCell.bind(this);
    }
    ngOnInit(): void {
        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.gridSettings = {
            columnWidth: 140,
        } as GridSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

HeaderCellInfo

The event `headerCellInfo` triggers while rendering the header cells in the pivot table. It allows the user to customize the element of the current header cell. It has the following parameters:

- `node` - It holds the current header cell info

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public columnGrandTotalIndex?: number;
  public rowGrandTotalIndex?: number;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotView;
  headerCell(args: any): void {
    ((this.pivotGridObj as PivotView).renderModule as
any).columnCellBoundEvent(args);
    //triggers for every cell
  }
  enginePopulated(args: any): void {
    ((this.pivotGridObj as PivotView).grid as Grid).headerCellInfo =
this.headerCell.bind(this);
  }
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      columnWidth: 140,
    } as GridSettings;
  }
}

```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ColumnRender

The event [columnRender](#) triggers while framing each columns for rendering in the pivot table. It allows the user to customize the text alignment, column visibility, autofit, re-ordering, minimum and maximum width for a specific column. It has the following parameters:

- **columns** - It holds the leaf level columns (i.e., value headers) information.
- **dataSourceSettings** - It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **name** - It holds the name of the event.
- **stackedColumns** - It holds the drilled columns (i.e., including column and value headers) information.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
import { Observable } from 'rxjs';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public observable = new Observable();
  ngOnInit(): void {
```

```

        this.width = '100%';
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.gridSettings = {
            columnWidth: 140,
            columnRender: this.observable.subscribe((args: any) => {
                // Here you can customize the specific columns.
                for (var i = 0; i < (args as any).columns.length; i++) {
                    (args as any).columns[i].autoFit = true;
                    (args as any).columns[i].textAlign="Right";
                }
            }) as any
        } as GridSettings;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

CellClick

The event [cellClick](#) triggers while clicking a cell in the pivot table. For instance, using this event end-user can either add or remove styles, edit value and also perform any other DOM manipulations. It has the following parameters:

- **currentCell** - It holds the current cell information.
- **data** - It holds the clicked cell's data like axis, formatted text, actual text, row header, column header and value informations.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, CellClickEventArgs } from '@syncfusion/ej2-
angular-pivotview';
import { noData } from './datasource';
@Component({
  imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width height='350'
(cellClick)='cellClick($event)'></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    public height?: number;
    cellClick(args: CellClickEventArgs) {
      //trigger for every cell click in pivot table
    }
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: noData as IDataset[],
        expandAll: true,
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        columns: [{ name: 'Date', showNoDataItems: true}],
        values: [{ name: 'Quantity', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country'}, { name: 'State'}],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Summary customization in Angular Pivotview component

Show or hide grand totals

Allows to show or hide grand totals in rows and columns using the [showGrandTotals](#) property. To hide the grand totals in rows and columns, set the property [showGrandTotals](#) in [dataSourceSettings](#) to **false**.

End user can also hide grand totals for row or columns separately by setting the property [showRowGrandTotals](#) or [showColumnGrandTotals](#) in [dataSourceSettings](#) to **false** respectively.

By default, [showGrandTotals](#), [showRowGrandTotals](#) and [showColumnGrandTotals](#) properties in [dataSourceSettings](#) are set as **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, FieldListService } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true'
width=width></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      showGrandTotals: false
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show Grand Totals at Top

Allows to show grand totals either at top or bottom in rows and columns using the [grandTotalsPosition](#) property. To show the grand totals at top in rows and columns, set the [grandTotalsPosition](#) property in [dataSourceSettings](#) to **Top**.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
width=width></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      grandTotalsPosition: 'Top',
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show or hide sub-totals

Allows to show or hide sub-totals in rows and columns using the [showSubTotals](#) property. To hide all the sub-totals in rows and columns, set the property [showSubTotals](#) in [dataSourceSettings](#) to **false**. End user can also hide sub-totals for rows or columns separately by setting the property [showRowSubTotals](#) or [showColumnSubTotals](#) in [dataSourceSettings](#) to **false** respectively.

By default, [showSubTotals](#), [showRowSubTotals](#) and [showColumnSubTotals](#) properties in [dataSourceSettings](#) are set as **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true' width=width></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      allowLabelFilter: true,
      allowValueFilter: true,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      showSubTotals: false
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show or hide sub-totals for specific fields

Allows to show or hide sub-totals for specific fields in rows and columns using the [ShowSubTotals](#) property. To hide sub-totals for a specific field in row or column axis, set the property [showSubTotals](#) in [rows](#) or [columns](#) to **false** respectively.

By default, [showSubTotals](#) property in [rows](#) or [columns](#) is set as **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, FieldListService } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true'
width=width></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year',
showSubTotals: false }, { name: 'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country', showSubTotals: false }, { name:
'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Show sub-totals at top or bottom

Allows to show sub-totals either at top or bottom of the header group in rows and columns by using the [subTotalsPosition](#) property. By default, [subTotalsPosition](#) property is set to **Auto**, which means that column sub-totals are displayed at the bottom and row sub-totals are displayed at the top of the header group in the pivot table.

To show sub-totals at top of the header group in rows and columns, set the [subTotalsPosition](#) property in [dataSourceSettings](#) to **Top**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
width=width></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      drilledMembers: [{ name: 'Country', items: ['France'] }, { name:
'Year', items: ['FY 2015'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      subTotalsPosition: 'Top',
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

To show sub-totals at bottom of the header group in rows and columns, set the [subTotalsPosition](#) property in [dataSourceSettings](#) to **Bottom**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
width=width></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      drilledMembers: [{ name: 'Country', items: ['France'] }, { name:
'Year', items: ['FY 2015'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      subTotalsPosition: 'Bottom',
    };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Show or hide totals using toolbar

It can also be achieved using built-in toolbar options by setting the [showToolbar](#) property in pivot table to **true**. Also, include the items **GrandTotal** and **SubTotal** within the [toolbar](#) property in pivot table. End user can now see "Show/Hide Grand totals" and "Show/Hide Sub totals" icons in toolbar UI automatically.

The grand totals and sub-totals can be dynamically displayed at the top or bottom of the pivot table's row and column axes by using the built-in options "Grand totals position" and "Subtotals position" available in the grand totals and sub-totals drop down menus, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  NumberFormattingService,
  ToolbarService, ConditionalFormattingService, ToolbarItems,
  DisplayOption, IDataset
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [CalculatedFieldService, ToolbarService,
  ConditionalFormattingService, FieldListService, NumberFormattingService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings width='100%'
showToolbar='true' height='350' [toolbar]='toolbarOptions' ></ejs-
pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public toolbarOptions?: ToolbarItems[];
  public displayOption?: DisplayOption;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.toolbarOptions = [ 'SubTotal', 'GrandTotal' ] as
ToolbarItems[];
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
```

```

        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hyper link in Angular Pivotview component

The pivot table supports to show hyperlink option to link data for individual cells that are displayed in the component. Also, the hyperlink can be enabled separately for row headers, column headers, value cells, and summary cells using the [hyperlinkSettings](#). It can be configured through code behind, during initial rendering and the settings available to show hyperlink are:

- [showHyperlink](#): It allows to set the visibility of hyperlink in all cells.
- [showRowHeaderHyperlink](#): It allows to set the visibility of hyperlink in row headers.
- [showColumnHeaderHyperlink](#): It allows to set the visibility of hyperlink in column headers.
- [showValueCellHyperlink](#): It allows to set the visibility of hyperlink in value cells.
- [showSummaryCellHyperlink](#): It allows to set the visibility of hyperlink in summary cells.
- [headerText](#): It allows to set the visibility of hyperlink based on header text.
- [conditionalSettings](#): It allows to set the visibility of hyperlink based on specific condition.

```
<!-- markdownlint-disable MD028 -->
```

By default, the hyperlink options are disabled for all cells in the pivot table.

User defined style can be applied to hyperlink using [cssClass](#) property in [hyperlinkSettings](#).

Hyperlink for all cells

The pivot table has an option to show hyperlink option for all cells that are currently in display. To do so, user need to set [showHyperlink](#) to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({

```

```

imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
}))
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public hyperlinkSettings?: HyperlinkSettings;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.hyperlinkSettings = {
            showHyperlink: true,
            cssClass: 'e-custom-class'
        } as HyperlinkSettings;
        this.width = '100%';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hyperlink for row headers

The pivot table has an option to show hyperlink option for row header cells alone that are currently in display. To do so, user need to set [showRowHeaderHyperlink](#) to true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';

```



```

import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public hyperlinkSettings?: HyperlinkSettings;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.hyperlinkSettings = {
      showRowHeaderHyperlink: true,
      cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hyperlink for column headers

The pivot table has an option to show hyperlink option for column header cells alone that are currently in display. To do so, user need to set [showColumnHeaderHyperlink](#) to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public hyperlinkSettings?: HyperlinkSettings;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.hyperlinkSettings = {
      showColumnHeaderHyperlink: true,
      cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hyperlink for value cells

The pivot table has an option to show hyperlink option for value cells alone that are currently in display. To do so, user need to set [showValueCellHyperlink](#) to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public hyperlinkSettings?: HyperlinkSettings;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.hyperlinkSettings = {
            showValueCellHyperlink: true,
            cssClass: 'e-custom-class'
        } as HyperlinkSettings;
        this.width = '100%';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hyperlink for summary cells

The pivot table has an option to show hyperlink option for summary cells alone that are currently in display. To do so, user need to set [showSummaryCellHyperlink](#) to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public hyperlinkSettings?: HyperlinkSettings;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.hyperlinkSettings = {
      showSummaryCellHyperlink: true,
      cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Condition based hyperlink

The pivot table has an option to show hyperlink in the cells based on specific conditions. It can be configured using the [conditionalSettings](#) through code behind, during initial rendering. The settings required are:

- [measure](#): Specifies the value field name, in-order to set the visibility of hyperlink for the same when condition is met.
- [conditions](#): Specifies the operator type such as **Equals**, **GreaterThan**, **LessThan**, etc.
- [value1](#): Specifies the start value.
- [value2](#): Specifies the end value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public hyperlinkSettings?: HyperlinkSettings;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.hyperlinkSettings = {
        conditionalSettings: [{
            measure: 'Sold',
            conditions: 'Between',
            value1: 150,
            value2: 200
        }],
        cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Header based hyperlink

The pivot table has an option to show hyperlink in the cells based on specific row or column header. It can be configured using the [headerText](#) option through code behind, during initial rendering.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public width?: string;

```

```

public dataSourceSettings?: IDataOptions;
public hyperlinkSettings?: HyperlinkSettings;
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Year', items: ['FY 2015'] }, { name:
'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.hyperlinkSettings = {
        headerText: 'FY 2015.Q1.Units Sold',
        cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Event

The event [hyperlinkCellClick](#) fires on every hyperlink cell click.

It has following parameters - **cancel** and **currentCell**. The parameter **currentCell** is used to customize the host cell element by any means. Meanwhile, when the parameter **cancel** is set to **true**, applied customization will not be updated to the host cell element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, HyperlinkSettings } from '@syncfusion/ej2-
angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],

```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[hyperlinkSettings]=hyperlinkSettings width=width
(hyperlinkCellClick)='hyperlinkCellClicked($event)'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public hyperlinkSettings?: HyperlinkSettings;
  hyperlinkCellClicked(args: any) {
    args.cancel = false;
    args.currentCell.setAttribute("data-url",
"https://ej2.syncfusion.com/");//here we have redirected to EJ2 Syncfusion
on hyperlinkcell click
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.hyperlinkSettings = {
      showRowHeaderHyperlink: true,
      cssClass: 'e-custom-class'
    } as HyperlinkSettings;
    this.width = '100%';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Apply condition based hyperlink for specific row or column](#)

Tool bar in Angular Pivotview component

Toolbar option allows to access the frequently used features like switching between pivot table and pivot chart, changing chart types, conditional formatting, exporting, etc... with ease at runtime. This

option can be enabled by setting the [showToolbar](#) property in pivot table to **true**. The [toolbar](#) property in pivot table accepts the collection of built-in toolbar options.

The following table shows built-in toolbar options and its actions.

Built-in Toolbar Options	Actions
----- -----	
New	Creates a new report
Save	Saves the current report
Save As	Save as current report
Rename	Renames the current report
Delete	Deletes the current report
Load	Loads any report from the report list
Grid	Shows pivot table
Chart	Shows a chart in any type from the built-in list and option to enable/disable multiple axes
Exporting	Exports the pivot table as PDF/Excel/CSV and the pivot chart as PDF and image
Sub-total	Shows or hides sub totals
Grand Total	Shows or hides grand totals
Conditional Formatting	Shows the conditional formatting pop-up to apply formatting
Number Formatting	Shows the number formatting pop-up to apply number formatting
Field List	Shows the fieldlist pop-up
MDX	Shows the MDX query that was run to retrieve data from the OLAP data source. NOTE: This applies only to the OLAP data source.

The order of toolbar options can be changed by simply moving the position of items in the **ToolbarItems** collection. Also if end user wants to remove any toolbar option from getting displayed, it can be simply ignored from adding into the **ToolbarItems** collection.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  NumberFormattingService,
  ToolbarService, ConditionalFormattingService, ToolbarItems,
  DisplayOption, IDataset, DataSourceSettings
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
```

```

        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [CalculatedFieldService, ToolbarService,
ConditionalFormattingService, FieldListService, NumberFormattingService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings
allowExcelExport='true' allowNumberFormatting='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolBar='true'
allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' height='350' [toolbar]='toolbarOptions'
(saveReport)='saveReport($event)' (loadReport)='loadReport($event)'
(fetchReport)='fetchReport($event)' (renameReport)='renameReport($event)'
(removeReport)='removeReport($event)' (newReport)='newReport()'></ejs-
pivotview></div>`
    ))
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    saveReport(args: any) {
        let reports = [];
        let isSaved: boolean = false;
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reports = JSON.parse(localStorage['pivotviewReports']);
        }
        if (args.report && args.reportName && args.reportName !== '') {
            reports.map(function (item: any): any {
                if (args.reportName === item.reportName) {
                    item.report = args.report; isSaved = true;
                }
            });
            if (!isSaved) {
                reports.push(args);
            }
            localStorage['pivotviewReports'] = JSON.stringify(reports);
        }
    }
    fetchReport(args: any) {
        let reportCollection: string[] = [];
        let reeportList: string[] = [];
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        reportCollection.map(function (item: any): void {
reeporList.push(item.reportName); });
        args.reportName = reeportList;
    }
    loadReport(args: any) {
        let reportCollection: string[] = [];

```

```

        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        reportCollection.map(function (item: any): void {
            if (args.reportName === item.reportName) {
                args.report = item.report;
            }
        });
        if (args.report) {
            (this.pivotGridObj!.dataSourceSettings as DataSourceSettings) =
JSON.parse(args.report).dataSourceSettings;
        }
    }
    removeReport(args: any) {
        let reportCollection: any[] = [];
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        for (let i: number = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.reportName) {
                reportCollection.splice(i, 1);
            }
        }
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
        }
    }
    renameReport(args: any) {
        let reportCollection: string[] = [];
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        reportCollection.map(function (item: any): any { if (args.reportName
=== item.reportName) { item.reportName = args.rename; } });
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
        }
    }
    newReport() {
        this.pivotGridObj?.setProperties({ dataSourceSettings: { columns:
[], rows: [], values: [], filters: [] } }, false);
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
        'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'] as ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],

```

```

        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show desired chart types in the dropdown menu

By default, all chart types are displayed in the dropdown menu included in the toolbar. However, based on the request for an application, we may need to show selective chart types on our own. This can be achieved using the [chartTypes](#) property. To know more about supporting chart types, [click here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, ToolbarService, ToolbarItems,
DisplayOption, IDataset
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ ToolbarService ],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings showToolbar='true'
width='100%' [displayOption]='displayOption' height='350'
[toolbar]='toolbarOptions' [chartTypes]= 'chartTypeOptions'></ejs-
pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;

```

```

public toolbarOptions?: ToolbarItems[];
public displayOption?: DisplayOption;
public chartTypeOptions?: any;
@ViewChild('pivotview', {static: false})
public pivotGridObj?: PivotView;
ngOnInit(): void {
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.chartTypeOptions = ['Column', 'Bar', 'Line', 'Area'];
    this.toolbarOptions = [ 'Grid', 'Chart' ] as ToolbarItems[];
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
    };
}
}

```

MAIN.TS

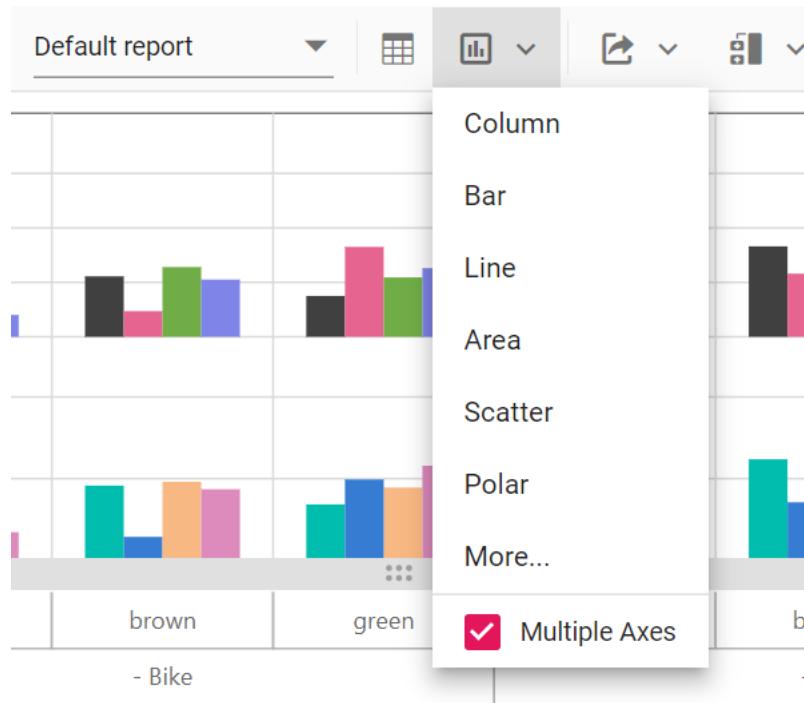
```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

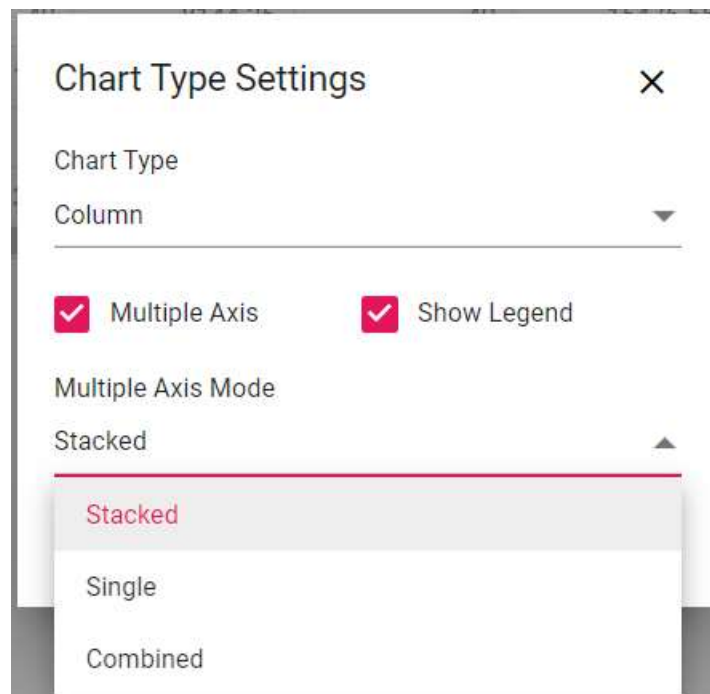
Switch the chart to multiple axes

In the chart, the user can switch from single axis to multiple axes with the help of the built-in checkbox available inside the chart type dropdown menu in the toolbar. For more information [refer here](#).



<!-- markdownlint-disable MD009 -->

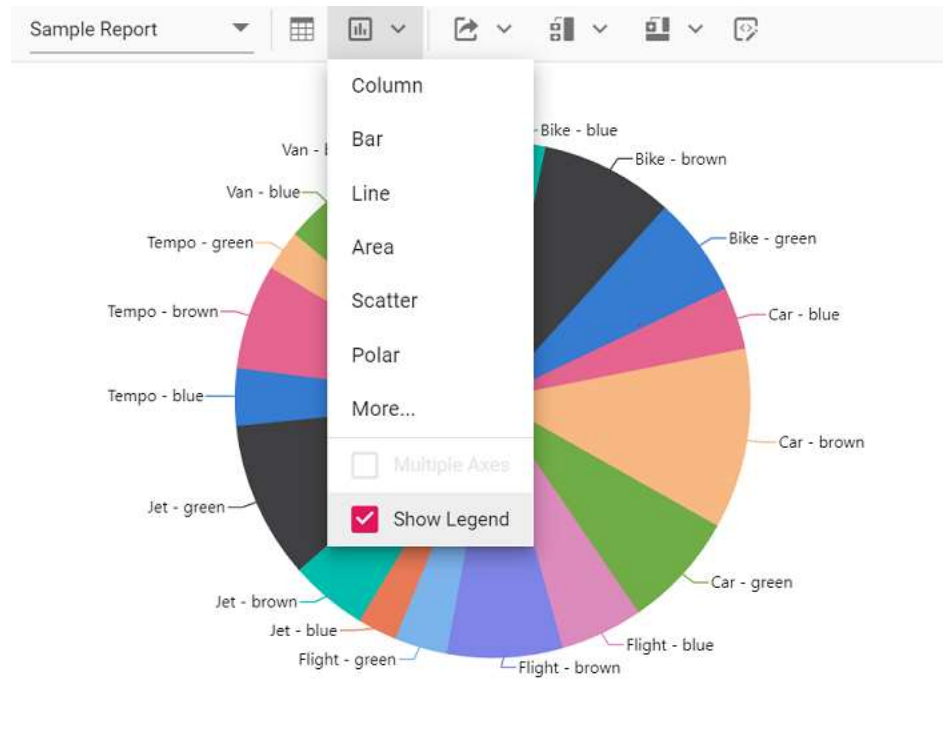
There are three modes available in **Multiple Axis** option: **Stacked**, **Single** and **Combined**. The modes can be changed using "Multiple Axis Mode" drop-down list which appears while clicking the **More...** option.



Show or hide legend

In the chart, legend can be shown or hidden dynamically with the help of the built-in option available in the chart type drop-down menu.

By default, the legend is not be visible for the accumulation chart types like pie, doughnut, pyramid, and funnel. Users can enable or disable using the built-in checkbox option.



Adding custom option to the toolbar

In addition to the existing built-in toolbar items, new toolbar item(s) may also be included. This can be achieved by using the [toolbarRender](#) event. The action of the new toolbar item(s) can also be defined within this event.

The new toolbar item(s) can be added to the desired position in the toolbar using the `splice` option.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, ToolbarService, ToolbarItems, DisplayOption,
  IDataset, DataSourceSettings
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ToolbarService],
  // specifies the template string for the pivot table component
```

```

    template: `<div><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings showToolbar='true' width='100%'
[displayOption]='displayOption' height='350' [toolbar]='toolbarOptions'
(toolbarRender)='beforeToolbarRender($event)'></ejs-pivotview></div>`,
    styleUrls: ['./app.component.css']
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    beforeToolbarRender(args: any) {
      args.customToolbar.splice(12, 0, {
        prefixIcon: 'e-tool-expand e-icons', tooltipText:
'Expand/Collapse',
        click: this.toolbarClicked.bind(this),
      });
    }
    toolbarClicked(args: any) {
      (this.pivotGridObj!.dataSourceSettings as
DataSourceSettings).expandAll =
!this.pivotGridObj?.dataSourceSettings.expandAll;
    }
    ngOnInit(): void {
      this.displayOption = { view: 'Both' } as DisplayOption;
      this.toolbarOptions = ['Expand/Collapse'] as any as ToolbarItems[];
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

In the above topic, we have seen how to add an icon as one of the toolbar item in toolbar panel. In the next topic, we are going to see how to frame the entire toolbar panel and how to add a custom control in it.

Toolbar Template

It allows to customize the toolbar panel by using template option. It allows any custom control to be used as one of the toolbar item inside the toolbar panel. It can be achieved by two ways,

Here, the entire toolbar panel can be framed in HTML elements that are appended at the top of the pivot table. The **id** of the HTML element needs to be set in the [toolbarTemplate](#) property in-order to map it to the pivot table.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, ToolbarService, IDataset } from
'@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { enableRipple } from '@syncfusion/ej2-base';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
enableRipple(false);
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ToolbarService],
  // specifies the template string for the pivot table component
  template: `<div class="control-section" id="pivot-table-section">
    <div>
      <ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings showToolbar='true' width='100%'
height='450' [gridSettings]='gridSettings'
[toolbarTemplate]='toolbarOptions'> </ejs-pivotview>
    </div>
    <div id='template'>
      <div>
        <div><button ej2-button id='expandall' class="e-flat"
(click)="enableRtl()">Expand All</button></div>
        <div><button ej2-button id='collapseall' class="e-flat"
(click)="disableRtl()">Collapse All</button></div>
      </div>
    </div></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public toolbarOptions?: string;
  public btnExpand?: Button;
  public btnCollapse?: Button;
  @ViewChild('pivotview')
  public pivotObj?: PivotView;
```

```

enableRTL() {
    (this.pivotObj as PivotView).dataSourceSettings.expandAll = true;
};
disableRTL(){
    (this.pivotObj as PivotView).dataSourceSettings.expandAll = false;
};
ngOnInit(): void {
    this.gridSettings = {
        columnWidth: 140
    } as GridSettings;
    this.toolbarOptions = '#template';
    this.dataSourceSettings = {
        enableSorting: true,
        columns: [{ name: 'Year' }, { name: 'Order_Source', caption:
'Order Source' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        values: [{ name: 'Sold', caption: 'Units Sold' },
        { name: 'Amount', caption: 'Sold Amount' }],
        filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
    };
    this.btnExpand = new Button({ isPrimary: true });
    this.btnExpand.appendTo('#expandall');
    this.btnCollapse = new Button({ isPrimary: true });
    this.btnCollapse.appendTo('#collapseall');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Another option allows to frame a custom toolbar item using HTML elements and include in the toolbar panel at the desired position. The custom toolbar items can be declared as control **instance** or element **id** in the [toolbar](#) property in pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import {
    IDataOptions, PivotView,
    ToolbarService, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';

```

```

import { enableRipple } from '@syncfusion/ej2-base';
import { Pivot_Data } from './datasource';
enableRipple(false);
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ToolbarService],
  // specifies the template string for the pivot table component
  template: `<div class="control-section" id="pivot-table-section">
    <div>
      <ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings showToolbar='true' width='100%'
height='450' [gridSettings]='gridSettings' [toolbar]='toolbarOptions'>
    </ejs-pivotview>
    </div>
    <ng-template #btnenablertl let-data>
      <button ejs-button id='enablertl' class="e-flat" isPrimary="true"
(click)="enableRtl()">Enable Rtl</button>
    </ng-template>
    <ng-template #btndisablertl let-data>
      <button ejs-button id='disablertl' class="e-flat" isPrimary="true"
(click)="disableRtl()">Disable Rtl</button>
    </ng-template></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public toolbarOptions?: any;
  @ViewChild('btnenablertl', {static:true})
  public enableRtl?: any;
  @ViewChild('btndisablertl', {static:true})
  public disableRTL?: any;
  @ViewChild('pivotview')
  public pivotObj?: PivotView;
  enablertl() {
    this.pivotObj!.enableRtl=true;
  };
  disableRtl(){
    this.pivotObj!.enableRtl=false;
  };
  ngOnInit(): void {
    this.gridSettings = {
      columnWidth: 140
    } as GridSettings;
    this.toolbarOptions = [{template: this.enableRtl},{template:
this.disableRTL}] as any;
    this.dataSourceSettings = {
      enableSorting: true,
      columns: [{ name: 'Year' }, { name: 'Order_Source', caption:
'Order Source' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],

```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        values: [{ name: 'Sold', caption: 'Units Sold' },
        { name: 'Amount', caption: 'Sold Amount' }],
        filters: [{ name: 'Product_Categories', caption: 'Product
Categories' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Note: For both options, the actions for the toolbar template items can be defined in the event [toolbarClick](#). Also, if the toolbar item is a custom control then its built-in events can also be accessed.

<!-- markdownlint-disable MD009 -->

Save and load report as a JSON file

The current pivot report can be saved as a JSON file in the desired path and loaded back to the pivot table at any time.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width showFieldList='true' (dataBound)='ondataBound($event)'></ejs-
pivotview></div><a id="save" class="btn btn-primary">Save</a><div
class="fileUpload btn btn-primary"><span>Load</span><input id="files"
type="file" class="upload" /></div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {

```

```

public dataSourceSettings?: IDataOptions;
public button?: Button;
@ViewChild('pivotview', {static: false})
public pivotGridObj?: PivotView;
public width?: string;
ondataBound(args: any) {
    var dataSource =
JSON.parse(this.pivotGridObj!.getPersistData()).dataSourceSettings.dataSource
e;

    var a = document.getElementById('save') as HTMLElement ;
    var mime_type = 'application/octet-stream'; // text/html, image/png,
et c

    a.setAttribute('download', 'pivot.JSON');
    (a as any).href = 'data:' + mime_type + ';base64,' +
btoa(JSON.stringify(dataSource) || '');
    (document.getElementById('files') as
HTMLElement).addEventListener('change', this.readBlob, false);
}
readBlob(args: any) {
    var files = (document.getElementById('load') as any).files;
    var file = files[0];
    var start = 0;
    var stop = file.size - 1;
    var reader = new FileReader();
    var $this = this;
    reader.onloadend = function(evt: any) {
        if (evt!.target.readyState == FileReader.DONE) {
            $this.pivotGridObj!.dataSourceSettings.dataSource =
JSON.parse(evt.target.result);
        }
    };
    var blob = file.slice(start, stop + 1);
    reader.readAsBinaryString(blob);
}
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

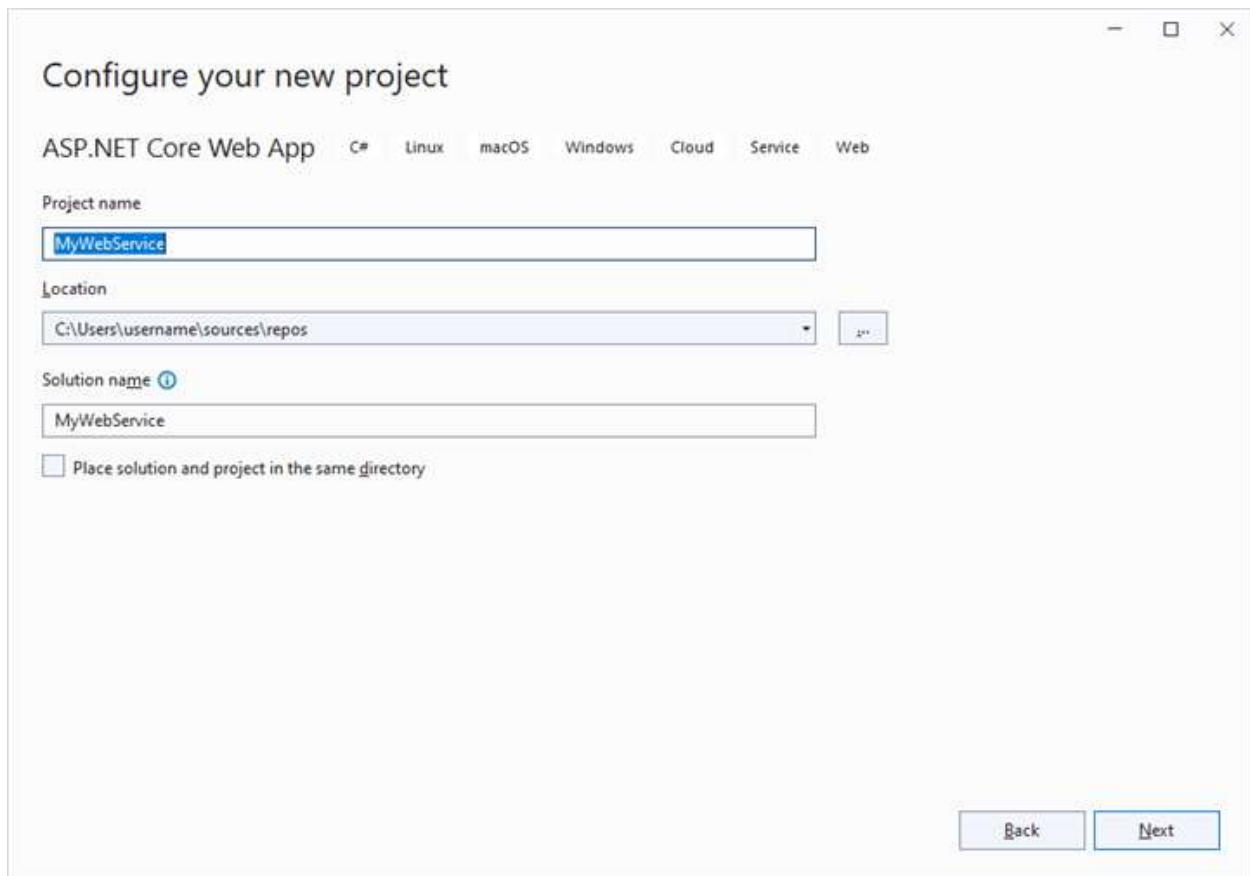
```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Save and load reports to a SQL database

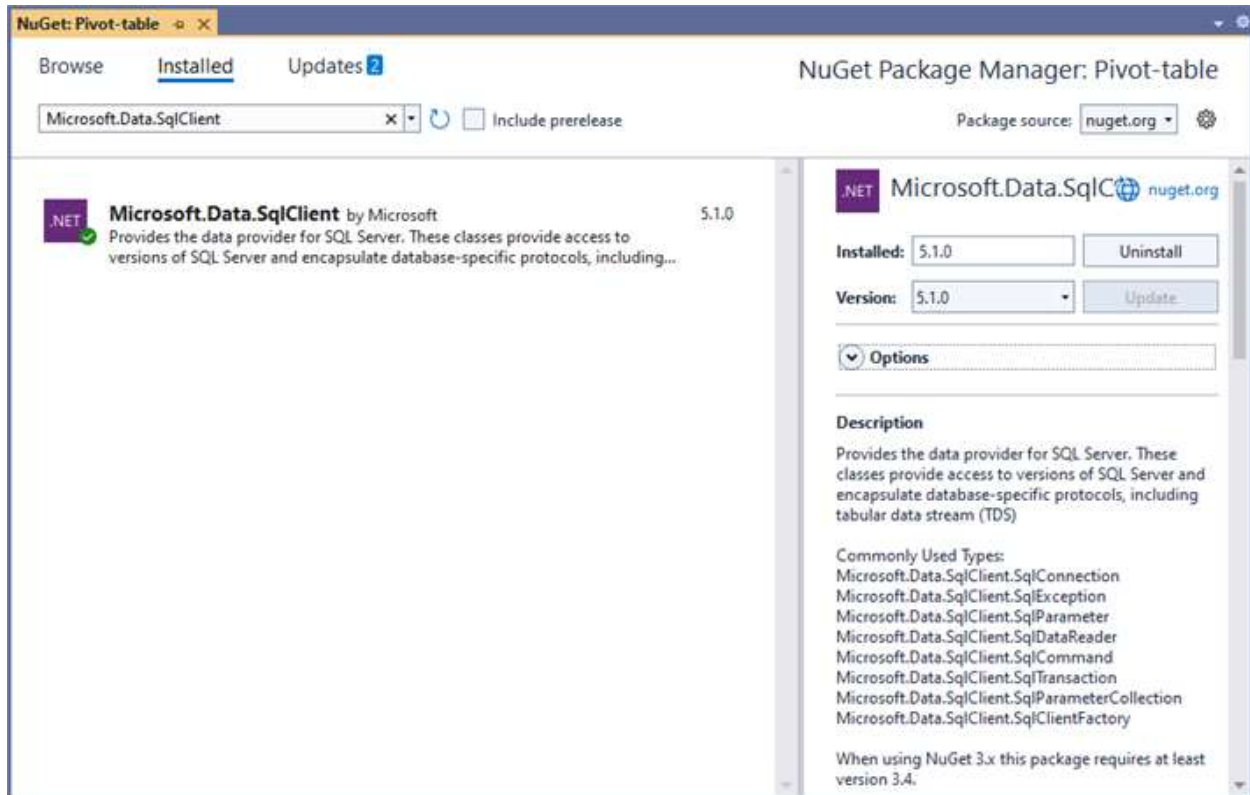
SQL Server is a relational database management system (RDBMS) that can be used to store and manage large amounts of data. In this topic, we will see how to save, save as, rename, load, delete, and add reports between a SQL Server database and a Angular Pivot Table at runtime.

Create a Web API service to connect to a SQL Server database

1. Open Visual Studio and create an ASP.NET Core Web App project type, naming it **MyWebService**. To create an ASP.NET Core Web application, follow the document [link](#).

The image shows the 'Configure your new project' window in Visual Studio. The 'ASP.NET Core Web App' template is selected. The 'Project name' field contains 'MyWebService'. The 'Location' field shows 'C:\Users\username\source\repos'. The 'Solution name' field also contains 'MyWebService'. There is an unchecked checkbox labeled 'Place solution and project in the same directory'. At the bottom right, there are 'Back' and 'Next' buttons.

2. To connect a SQL Server database using the Microsoft SqlClient in our application, we need to install the [Microsoft.Data.SqlClient](#) NuGet package. To do so, open the NuGet package manager of the project solution, search for the package **Microsoft.Data.SqlClient** and install it.



3. Under the **Controllers** folder, create a Web API controller (aka, PivotController.cs) file that aids in data communication with the Pivot Table.

4. In the Web API Controller (aka, PivotController), the **OpenConnection** method is used to connect to the SQL database. The **GetDataTable** method then processes the specified SQL query string, retrieves data from the database, and converts it into a **DataTable** using **SqlCommand** and **SqldataAdapter**. This **DataTable** can be used to retrieve saved reports and modify them further as shown in the code block below.

[PivotController.cs]

```
`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebService.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpPost]
```

```
[Route("Pivot/SaveReport")]
public void SaveReport([FromBody] Dictionary<string, string> reportArgs)
{
    SaveReportToDB(reportArgs["reportName"], reportArgs["report"]);
}

[HttpPost]
[Route("Pivot/FetchReport")]
public IActionResult FetchReport()
{
    return Ok((FetchReportListFromDB()));
}

[HttpPost]
[Route("Pivot/LoadReport")]
public IActionResult LoadReport([FromBody] Dictionary<string, string> reportArgs)
{
    return Ok((LoadReportFromDB(reportArgs["reportName"])));
}

[HttpPost]
[Route("Pivot/RemoveReport")]
public void RemoveReport([FromBody] Dictionary<string, string> reportArgs)
{
    RemoveReportFromDB(reportArgs["reportName"]);
}

[HttpPost]
[Route("Pivot/RenameReport")]
public void RenameReport([FromBody] RenameReportDB reportArgs)
{
    RenameReportInDB(reportArgs.ReportName, reportArgs.RenameReport, reportArgs.isReportExists);
}

public class RenameReportDB
{
    public string ReportName { get; set; }
    public string RenameReport { get; set; }
```



```
public bool isReportExists { get; set; }
}

private void SaveReportToDB(string reportName, string report)
{
    SqlConnection sqlConn = OpenConnection();
    bool isDuplicate = true;
    SqlCommand cmd1 = null;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {
            isDuplicate = false;
            cmd1 = new SqlCommand("update ReportTable set Report=@Report where ReportName like
@ReportName", sqlConn);
        }
    }
    if (isDuplicate)
    {
        cmd1 = new SqlCommand("insert into ReportTable (ReportName,Report)
Values(@ReportName,@Report)", sqlConn);
    }
    cmd1.Parameters.AddWithValue("@ReportName", reportName);
    cmd1.Parameters.AddWithValue("@Report", report.ToString());
    cmd1.ExecuteNonQuery();
    sqlConn.Close();
}

private string LoadReportFromDB(string reportName)
{
    SqlConnection sqlConn = OpenConnection();
    string report = string.Empty;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {

```

```
report = (string)row["Report"];
break;
}
}
sqlConn.Close();
return report;
}
private List<string> FetchReportListFromDB()
{
    SqlConnection sqlConn = OpenConnection();
    List<string> reportNames = new List<string>();
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if (!string.IsNullOrEmpty(row["ReportName"] as string))
        {
            reportNames.Add(row["ReportName"].ToString());
        }
    }
    sqlConn.Close();
    return reportNames;
}
private void RenameReportInDB(string reportName, string renameReport, bool isReportExists)
{
    SqlConnection sqlConn = OpenConnection();
    SqlCommand cmd1 = null;
    if (isReportExists)
    {
        foreach (DataRow row in GetDataTable(sqlConn).Rows)
        {
            if ((row["ReportName"] as string).Equals(reportName))
            {
                cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
                    sqlConn);
            }
        }
    }
}
```

```
break;
}
}
cmd1.ExecuteNonQuery();
}
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("update ReportTable set ReportName=@RenameReport where ReportName
like '%" + reportName + "%'", sqlConn);
break;
}
}
cmd1.Parameters.AddWithValue("@RenameReport", renameReport);
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private void RemoveReportFromDB(string reportName)
{
SqlConnection sqlConn = OpenConnection();
SqlCommand cmd1 = null;
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
sqlConn);
break;
}
}
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
```

```

private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}

private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
}

```

5. When you run the app, it will be hosted at <https://localhost:44313>. You can use the hosted URL to save and load reports in the SQL database from the Pivot Table.

Further, let us explore more on how to save, load, rename, delete, and add reports using the built-in toolbar options via Web API controller (aka, PivotController) one-by-one.

[Saving a report](#)

When you select the **“Save a report”** option from the toolbar, the [saveReport](#) event is triggered. In this event, an AJAX request is made to the Web API controller's **SaveReport** method, passing the name of the current report and the current report, which you can use to check and save in the SQL database.

For example, the report shown in the following code snippet will be passed to the **SaveReport** method along with the report name **“Sample Report”** and saved in the SQL database.

```

[app.component.ts]
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, FieldListService, CalculatedFieldService,

```

```

ToolbarService, ConditionalFormattingService, ToolbarItems, DisplayOption, IDataset,
NumberFormattingService,
FetchReportArgs,
LoadReportArgs,
RemoveReportArgs,
RenameReportArgs,
SaveReportArgs
} from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
import { enableRipple } from '@syncfusion/ej2-base';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartsettings';
enableRipple(false);
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  providers: [CalculatedFieldService, ToolbarService, ConditionalFormattingService, FieldListService,
    NumberFormattingService],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions | undefined;
  public gridSettings: GridSettings | undefined;
  public toolbarOptions: ToolbarItems[] | undefined;
  public chartSettings: ChartSettings | undefined;
  public displayOption: DisplayOption | undefined;
  @ViewChild('pivotview')
  public pivotTableObj: PivotView | undefined;
  saveReport(args: SaveReportArgs) {
    var report = JSON.parse(args.report as string);
    report.dataSourceSettings.dataSource = [];
    fetch('https://localhost:44313/Pivot/SaveReport', {
      method: 'POST',
      headers: {

```

```

'Accept': 'application/json',
'Content-Type': 'application/json',
},
body: JSON.stringify({ reportName: args.reportName, report: JSON.stringify(report) })
}).then(response => {
this.fetchReport(args as any);
});
}
ngOnInit(): void {
this.chartSettings = {
chartSeries: { type: 'Column', animation: { enable: false } },
enableMultipleAxis: false, value: 'Amount', enableExport: true
} as ChartSettings;
this.displayOption = { view: 'Both' } as DisplayOption;
this.gridSettings = {
columnWidth: 140
} as GridSettings;
this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
'Grid', 'Chart', 'MDX', 'Export', 'SubTotal', 'GrandTotal', 'ConditionalFormatting', 'FieldList'] as
ToolbarItems[];
this.dataSourceSettings = {
columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
dataSource: this.getPivotData(),
expandAll: false,
filters: [],
formatSettings: [{ name: 'Amount', format: 'C0' }],
rows: [{ name: 'Country' }, { name: 'Products' }],
values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
};
}
}
,
[PivotController.cs]

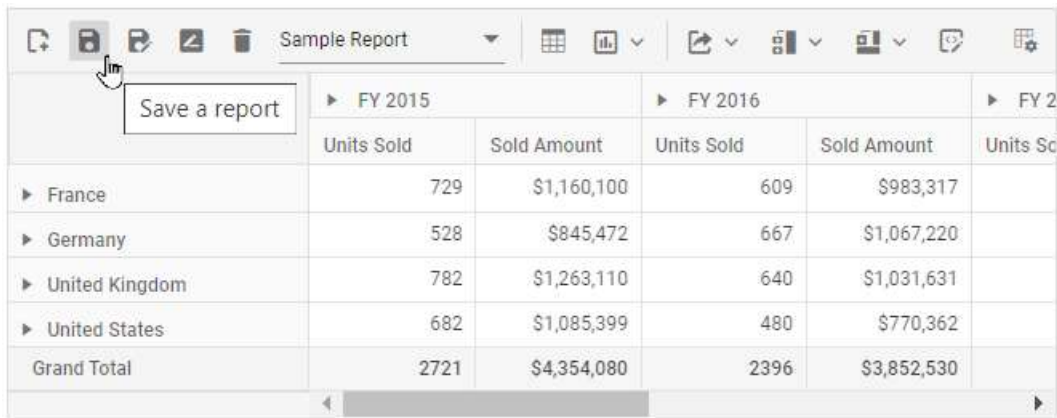
```

```
`csharp
namespace MyWebApp.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class PivotController : ControllerBase
    {
        [HttpPost]
        [Route("Pivot/SaveReport")]
        public void SaveReport([FromBody] Dictionary<string, string> reportArgs)
        {
            SaveReportToDB(reportArgs["reportName"], reportArgs["report"]);
        }
        private void SaveReportToDB(string reportName, string report)
        {
            SqlConnection sqlConn = OpenConnection();
            bool isDuplicate = true;
            SqlCommand cmd1 = null;
            foreach (DataRow row in GetDataTable(sqlConn).Rows)
            {
                if ((row["ReportName"] as string).Equals(reportName))
                {
                    isDuplicate = false;
                    cmd1 = new SqlCommand("update ReportTable set Report=@Report where ReportName like @ReportName", sqlConn);
                }
            }
            if (isDuplicate)
            {
                cmd1 = new SqlCommand("insert into ReportTable (ReportName,Report) Values(@ReportName,@Report)", sqlConn);
            }
            cmd1.Parameters.AddWithValue("@ReportName", reportName);
            cmd1.Parameters.AddWithValue("@Report", report.ToString());
        }
    }
}
```

```

cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);
    SqlDataAdapter da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
}
}
,

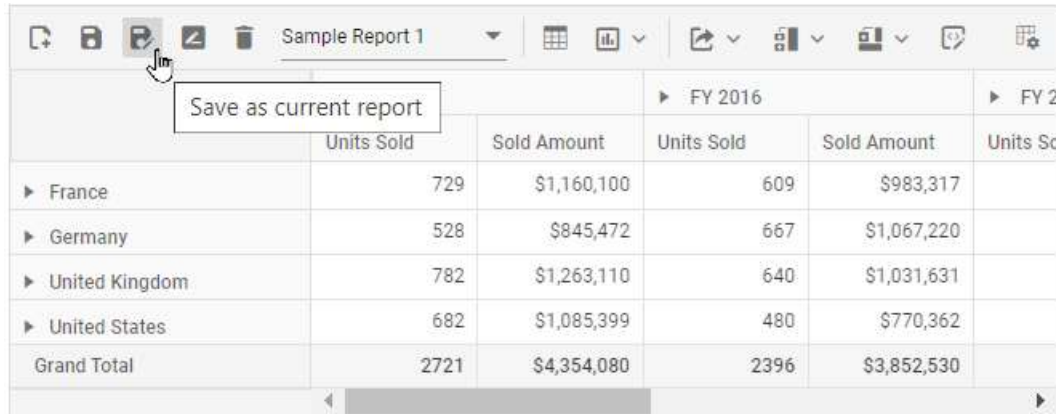
```



	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

In the meantime, you can save a duplicate of the current report to the SQL Server database with a different name by selecting **“Save as current report”** from the toolbar. The [saveReport](#) event will then

be triggered with the new report name “**Sample Report 1**” and the current report. You can save them to the SQL Server database after passing them to the Web API service, as mentioned above.



	Units Sold	Sold Amount	FY 2016 Units Sold	FY 2016 Sold Amount	FY 2017 Units Sold	FY 2017 Sold Amount
France	729	\$1,160,100	609	\$983,317		
Germany	528	\$845,472	667	\$1,067,220		
United Kingdom	782	\$1,263,110	640	\$1,031,631		
United States	682	\$1,085,399	480	\$770,362		
Grand Total	2721	\$4,354,080	2396	\$3,852,530		

Loading a report

When you select the dropdown menu item from the toolbar, the [loadReport](#) event is triggered. In this event, an AJAX request is made to the **LoadReport** method of the Web API controller, passing the name of the selected report. The method uses this information to search for the report in the SQL database, fetch it, and load it into the pivot table.

For example, if the report name “**Sample Report 1**” is selected from a dropdown menu and passed, the **LoadReport** method will use that name to search for the report in the SQL database, retrieve it, and then load it into the pivot table.

[app.component.ts]

`typescript

```
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  ToolbarService, ConditionalFormattingService, ToolbarItems, DisplayOption, IDataset,
  NumberFormattingService,
  FetchReportArgs,
  LoadReportArgs,
  RemoveReportArgs,
  RenameReportArgs,
  SaveReportArgs
} from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
import { enableRipple } from '@syncfusion/ej2-base';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartsettings';
enableRipple(false);
```

```
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  providers: [CalculatedFieldService, ToolbarService, ConditionalFormattingService, FieldListService,
    NumberFormattingService],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions | undefined;
  public gridSettings: GridSettings | undefined;
  public toolbarOptions: ToolbarItems[] | undefined;
  public chartSettings: ChartSettings | undefined;
  public displayOption: DisplayOption | undefined;
  @ViewChild('pivotview')
  public pivotTableObj: PivotView | undefined;
  loadReport(args: LoadReportArgs) {
    fetch('https://localhost:44313/Pivot/LoadReport', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ reportName: args.reportName })
    }).then(res => res.json())
    .then(response => {
      if (response) {
        var report = JSON.parse(response);
        report.dataSourceSettings.dataSource = (this.pivotTableObj as
          PivotView).dataSourceSettings.dataSource;
        (this.pivotTableObj as PivotView).dataSourceSettings = report.dataSourceSettings;
      }
    });
  }
  ngOnInit(): void {
```

```

this.chartSettings = {
  chartSeries: { type: 'Column', animation: { enable: false } },
  enableMultipleAxis: false, value: 'Amount', enableExport: true
} as ChartSettings;
this.displayOption = { view: 'Both' } as DisplayOption;
this.gridSettings = {
  columnWidth: 140
} as GridSettings;
this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
  'Grid', 'Chart', 'MDX', 'Export', 'SubTotal', 'GrandTotal', 'ConditionalFormatting', 'FieldList'] as
  ToolbarItems[];
this.dataSourceSettings = {
  columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
  dataSource: this.getPivotData(),
  expandAll: false,
  filters: [],
  formatSettings: [{ name: 'Amount', format: 'C0' }],
  rows: [{ name: 'Country' }, { name: 'Products' }],
  values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
};
}
}
,

```

```
[PivotController.cs]
```

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebApp.Controllers
{
  [ApiController]
  [Route("[controller]")]
  public class PivotController : ControllerBase

```

```

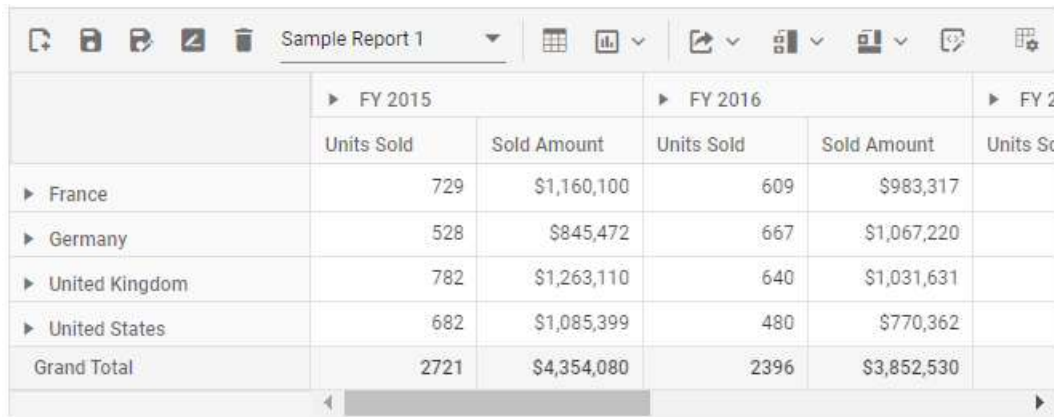
{
[HttpPost]
[Route("Pivot/LoadReport")]
public IActionResult LoadReport([FromBody] Dictionary<string, string> reportArgs)
{
return Ok((LoadReportFromDB(reportArgs["reportName"])));
}
private string LoadReportFromDB(string reportName)
{
SqlConnection sqlConn = OpenConnection();
string report = string.Empty;
foreach (DataRow row in GetDataTable(sqlConn).Rows)
{
if ((row["ReportName"] as string).Equals(reportName))
{
report = (string)row["Report"];
break;
}
}
sqlConn.Close();
return report;
}
private SqlConnection OpenConnection()
{
// Replace with your own connection string.
string connectionString = @"<Enter your valid connection string here>";
SqlConnection sqlConn = new SqlConnection(connectionString);
sqlConn.Open();
return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
string xquery = "select * from ReportTable";

```

```

SqlCommand cmd = new SqlCommand(xquery, sqlConn);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
return dt;
}
}
}
`

```



	FY 2015		FY 2016		FY 2017
	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Renaming a report

When you select the **“Rename a current report”** option from the toolbar, the [renameReport](#) event is triggered. In this event, an AJAX request is made to the **RenameReport** method of the Web API controller, passing the current and new report names, where you can use the current report name to identify the report and resave it with the new report name in the SQL database.

For example, if we rename the current report from **“Sample Report 1”** to **“Sample Report 2”**, both **“Sample Report 1”** and **“Sample Report 2”** will be passed to the **RenameReport** method, which will rename the current report with the new report name **“Sample Report 2”** in the SQL database.

[app.component.ts]

`typescript

```

import { Component, OnInit, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  ToolbarService, ConditionalFormattingService, ToolbarItems, DisplayOption, IDataset,
  NumberFormattingService,
  FetchReportArgs,
  LoadReportArgs,
  RemoveReportArgs,

```

```

RenameReportArgs,
SaveReportArgs
} from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
import { enableRipple } from '@syncfusion/ej2-base';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartsettings';
enableRipple(false);
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  providers: [CalculatedFieldService, ToolbarService, ConditionalFormattingService, FieldListService,
    NumberFormattingService],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions | undefined;
  public gridSettings: GridSettings | undefined;
  public toolbarOptions: ToolbarItems[] | undefined;
  public chartSettings: ChartSettings | undefined;
  public displayOption: DisplayOption | undefined;
  @ViewChild('pivotview')
  public pivotTableObj: PivotView | undefined;
  renameReport(args: RenameReportArgs) {
    fetch('https://localhost:44313/Pivot/RenameReport', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ reportName: args.reportName, renameReport: args.rename, isReportExists:
        args.isReportExists })
    }).then(response => {
      this.fetchReport(args as any);
    });
  }
}

```

```

}
ngOnInit(): void {
  this.chartSettings = {
    chartSeries: { type: 'Column', animation: { enable: false } },
    enableMultipleAxis: false, value: 'Amount', enableExport: true
  } as ChartSettings;
  this.displayOption = { view: 'Both' } as DisplayOption;
  this.gridSettings = {
    columnWidth: 140
  } as GridSettings;
  this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
    'Grid', 'Chart', 'MDX', 'Export', 'SubTotal', 'GrandTotal', 'ConditionalFormatting', 'FieldList'] as
    ToolbarItems[];
  this.dataSourceSettings = {
    columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
    dataSource: this.getPivotData(),
    expandAll: false,
    filters: [],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
  };
}

```

```
[PivotController.cs]
```

```
`csharp
```

```
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.Data.SqlClient;
```

```
using System.Data;
```

```
namespace MyWebApp.Controllers
```

```
{
```

```
[ApiController]
```

```

[Route("[controller]")]
public class PivotController : ControllerBase
{
    [HttpPost]
    [Route("Pivot/RenameReport")]
    public void RenameReport([FromBody] RenameReportDB reportArgs)
    {
        RenameReportInDB(reportArgs.ReportName, reportArgs.RenameReport, reportArgs.isReportExists);
    }
    public class RenameReportDB
    {
        public string ReportName { get; set; }
        public string RenameReport { get; set; }
        public bool isReportExists { get; set; }
    }
    private void RenameReportInDB(string reportName, string renameReport, bool isReportExists)
    {
        SqlConnection sqlConn = OpenConnection();
        SqlCommand cmd1 = null;
        if (isReportExists)
        {
            foreach (DataRow row in GetDataTable(sqlConn).Rows)
            {
                if ((row["ReportName"] as string).Equals(reportName))
                {
                    cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
                    sqlConn);
                    break;
                }
            }
            cmd1.ExecuteNonQuery();
        }
        foreach (DataRow row in GetDataTable(sqlConn).Rows)

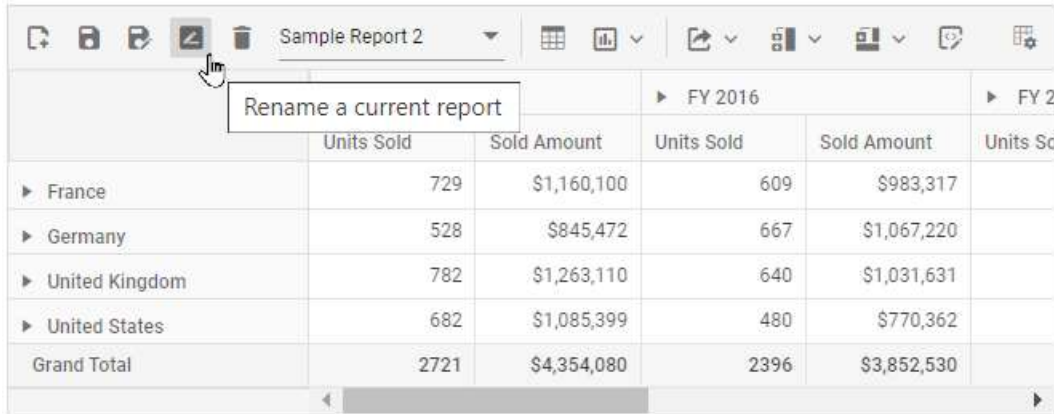
```



```

{
if ((row["ReportName"] as string).Equals(reportName))
{
cmd1 = new SqlCommand("update ReportTable set ReportName=@RenameReport where ReportName
like '%" + reportName + "%'", sqlConn);
break;
}
}
cmd1.Parameters.AddWithValue("@RenameReport", renameReport);
cmd1.ExecuteNonQuery();
sqlConn.Close();
}
private SqlConnection OpenConnection()
{
// Replace with your own connection string.
string connectionString = @"<Enter your valid connection string here>";
SqlConnection sqlConn = new SqlConnection(connectionString);
sqlConn.Open();
return sqlConn;
}
private DataTable GetDataTable(SqlConnection sqlConn)
{
string xquery = "select * from ReportTable";
SqlCommand cmd = new SqlCommand(xquery, sqlConn);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
return dt;
}
}
}

```



	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Deleting a report

When you select the **“Delete a current report”** option from the toolbar, the [removeReport](#) event is triggered. In this event, an AJAX request is made to the **RemoveReport** method of the Web API controller, passing the current report name to identify and delete the appropriate report from the SQL database.

Note: * If the current report **n** from the pivot table is deleted, the pivot table will automatically load the last report from the report list.

* When a report is removed from a pivot table with only one report, the SQL database refreshes; however, the pivot table will continue to show the removed report until a new report is added to the pivot table.

For example, if we delete the current report **“Sample Report 2”** from the pivot table, the current report name **“Sample Report 2”** is passed to the **RemoveReport** method, which allows you to identify and delete the report from the SQL database.

[app.component.ts]

`typescript

```
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  ToolbarService, ConditionalFormattingService, ToolbarItems, DisplayOption, IDataset,
  NumberFormattingService,
  FetchReportArgs,
  LoadReportArgs,
  RemoveReportArgs,
  RenameReportArgs,
  SaveReportArgs
} from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/gridsettings';
```

```
import { enableRipple } from '@syncfusion/ej2-base';
import { ChartSettings } from '@syncfusion/ej2-pivotview/src/pivotview/model/chartsettings';
enableRipple(false);
@Component({
  selector: 'app-root',
  // specifies the template string for the pivot table component
  providers: [CalculatedFieldService, ToolbarService, ConditionalFormattingService, FieldListService,
    NumberFormattingService],
  templateUrl: './app.component.html'
})
export class AppComponent implements OnInit {
  public dataSourceSettings: IDataOptions | undefined;
  public gridSettings: GridSettings | undefined;
  public toolbarOptions: ToolbarItems[] | undefined;
  public chartSettings: ChartSettings | undefined;
  public displayOption: DisplayOption | undefined;
  @ViewChild('pivotview')
  public pivotTableObj: PivotView | undefined;
  removeReport(args: RemoveReportArgs): void {
    fetch('https://localhost:44313/Pivot/RemoveReport', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ reportName: args.reportName })
    }).then(response => {
      this.fetchReport(args as any);
    });
  }
  ngOnInit(): void {
    this.chartSettings = {
      chartSeries: { type: 'Column', animation: { enable: false } },
    }
  }
}
```

```

enableMultipleAxis: false, value: 'Amount', enableExport: true
} as ChartSettings;
this.displayOption = { view: 'Both' } as DisplayOption;
this.gridSettings = {
columnWidth: 140
} as GridSettings;
this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove', 'Load',
'Grid', 'Chart', 'MDX', 'Export', 'SubTotal', 'GrandTotal', 'ConditionalFormatting', 'FieldList'] as
ToolbarItems[];
this.dataSourceSettings = {
columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
dataSource: this.getPivotData(),
expandAll: false,
filters: [],
formatSettings: [{ name: 'Amount', format: 'C0' }],
rows: [{ name: 'Country' }, { name: 'Products' }],
values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }]
};
}
}
`

```

[PivotController.cs]

```

`csharp
using Microsoft.AspNetCore.Mvc;
using Microsoft.Data.SqlClient;
using System.Data;
namespace MyWebApp.Controllers
{
[ApiController]
[Route("[controller]")]
public class PivotController : ControllerBase
{
[HttpPost]

```

```

[Route("Pivot/RemoveReport")]
public void RemoveReport([FromBody] Dictionary<string, string> reportArgs)
{
    RemoveReportFromDB(reportArgs["reportName"]);
}

private void RemoveReportFromDB(string reportName)
{
    SqlConnection sqlConn = OpenConnection();
    SqlCommand cmd1 = null;
    foreach (DataRow row in GetDataTable(sqlConn).Rows)
    {
        if ((row["ReportName"] as string).Equals(reportName))
        {
            cmd1 = new SqlCommand("delete from ReportTable where ReportName like '%" + reportName + "%'",
            sqlConn);
            break;
        }
    }
    cmd1.ExecuteNonQuery();
    sqlConn.Close();
}

private SqlConnection OpenConnection()
{
    // Replace with your own connection string.
    string connectionString = @"<Enter your valid connection string here>";
    SqlConnection sqlConn = new SqlConnection(connectionString);
    sqlConn.Open();
    return sqlConn;
}

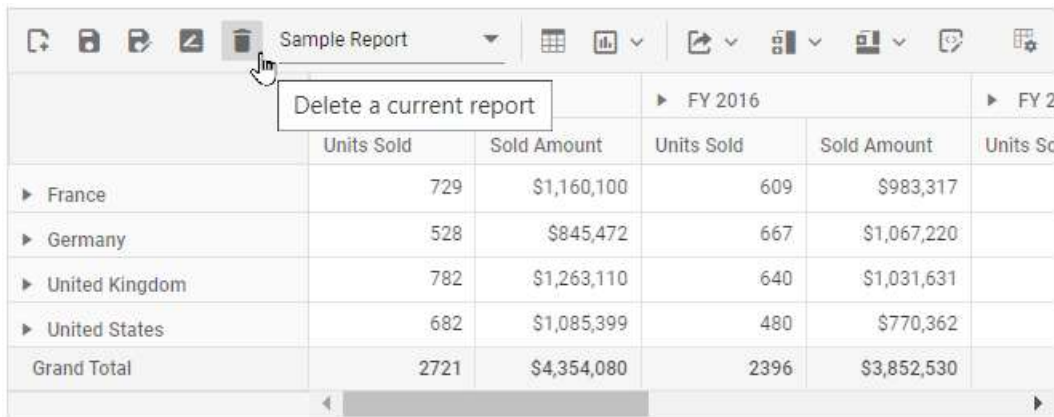
private DataTable GetDataTable(SqlConnection sqlConn)
{
    string xquery = "select * from ReportTable";
    SqlCommand cmd = new SqlCommand(xquery, sqlConn);

```

```

SqlDataAdapter da = new SqlDataAdapter(cmd);
DataTable dt = new DataTable();
da.Fill(dt);
return dt;
}
}
}
,

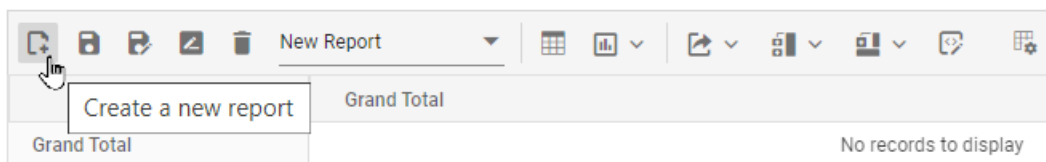
```



	Units Sold	Sold Amount	Units Sold	Sold Amount	Units Sold
France	729	\$1,160,100	609	\$983,317	
Germany	528	\$845,472	667	\$1,067,220	
United Kingdom	782	\$1,263,110	640	\$1,031,631	
United States	682	\$1,085,399	480	\$770,362	
Grand Total	2721	\$4,354,080	2396	\$3,852,530	

Adding a report

When you select the **“Create a new report”** option from the toolbar, the [newReport](#) event is triggered, followed by the [saveReport](#) event. To save this new report to the SQL database, use the [saveReport](#) event triggered later, and then follow the save report briefing in the preceding [topic](#).



	Grand Total
Grand Total	No records to display

Limitations with respect to report manipulation

Below points need to be considered when saving the report to SQL Server database.

- **Data source:** Both raw data and aggregated data won't be saved and loaded from the database.
- **Hyperlinks:** Option to link external facts via pivot table cells won't be saved and loaded from the database.
- The pivot table should always load reports from the SQL database based on the data source that is currently bound to it.

In [this](#) GitHub repository, you can find our Angular Pivot Table sample and ASP.NET Core Web Application to save and load reports from SQL Server database.

Events

FetchReport

The event [fetchReport](#) is triggered when dropdown list is clicked in the toolbar in-order to retrieve and populate saved reports. It has following parameter - `reportName`. This event allows user to fetch the report names from local storage and populate the dropdown list.

LoadReport

The event [loadReport](#) is triggered when a report is selected from the dropdown list in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to load the selected report to the pivot table.

NewReport

The event [newReport](#) is triggered when the new report icon is clicked in the toolbar. It has following parameter - `report`. This event allows user to create new report and add to the report list.

RenameReport

The event [renameReport](#) is triggered when rename report icon is clicked in the toolbar. It has following parameters - `rename`, `report` and `reportName`. This event allows user to rename the selected report from the report list.

RemoveReport

The event [removeReport](#) is triggered when remove report icon is clicked in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to remove the selected report from the report list.

SaveReport

The event [saveReport](#) is triggered when save report icon is clicked in the toolbar. It has following parameters - `report` and `reportName`. This event allows user to save the altered report to the report list.

<!-- markdownlint-disable MD009 -->

ToolbarRender

The [toolbarRender](#) event is triggered when the toolbar is rendered. It has the `customToolbar` parameter. This event helps to customize the built-in toolbar items and to [include new toolbar item\(s\)](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, ToolbarService, ToolbarItems, DisplayOption,
    IDataset, FieldListService
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
```

```

standalone: true,
  selector: 'app-container',
  providers: [ToolbarService, FieldListService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings showToolbar='true'
showFieldList='true' width='100%' [displayOption]='displayOption'
height='350' [toolbar]='toolbarOptions'
(toolbarRender)='beforeToolbarRender($event)'
(saveReport)='saveReport($event)'></ejs-pivotview></div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public toolbarOptions?: ToolbarItems[];
  public displayOption?: DisplayOption;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  saveReport(args: any) {
    let reports = [];
    let isSaved: boolean = false;
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
      reports = JSON.parse(localStorage['pivotviewReports']);
    }
    if (args.report && args.reportName && args.reportName !== '') {
      reports.map(function (item: any): any {
        if (args.reportName === item.reportName) {
          item.report = args.report; isSaved = true;
        }
      });
      if (!isSaved) {
        reports.push(args);
      }
      localStorage['pivotviewReports'] = JSON.stringify(reports);
    }
  }
  beforeToolbarRender(args: any) {
    args.customToolbar.splice(2, 0, {
      prefixIcon: 'e-rename-report e-icons', tooltipText: 'Custom
Button',
      click: this.customButton.bind(this),
    });
    args.customToolbar.splice(3, 0, {
      prefixIcon: 'e-tool-expand e-icons', tooltipText:
'Expand/Collapse',
      click: this.toolbarClicked.bind(this),
    });
    args.customToolbar[0].align = "Left";
    args.customToolbar[1].align = "Center";
    args.customToolbar[2].align = "Right";
  }
  customButton(args: any) {
    // Here you can customize the click event for custom button
  }
  toolbarClicked(args: any) {

```



```

        this.pivotGridObj!.dataSourceSettings.expandAll =
!this.pivotGridObj!.dataSourceSettings.expandAll;
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['Save', 'Export', 'FieldList'] as
ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

BeforeExport

The pivot table (or) pivot chart can be exported as a pdf, excel, csv etc., document using the toolbar options. And, you can customize the export settings for exporting document by using the [beforeExport](#) event in the toolbar.

For example, you can add the header and footer for the pdf document by setting the **header** and **footer** properties for the **pdfExportProperties** in the [beforeExport](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, FieldListService, CalculatedFieldService,
    ToolbarService, ConditionalFormattingService, ToolbarItems,
    DisplayOption, IDataset
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ]
})

```

```

    ],
    standalone: true,
    selector: 'app-container',
    providers: [CalculatedFieldService, ToolbarService,
ConditionalFormattingService, FieldListService],
    // specifies the template string for the pivot table component
    template: `<div><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true'
allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' height='350' [toolbar]='toolbarOptions'
(saveReport)='saveReport($event)' (loadReport)='loadReport($event)'
(fetchReport)='fetchReport($event)' (renameReport)='renameReport($event)'
(removeReport)='removeReport($event)' (newReport)='newReport()'
(beforeExport)='beforeExport($event)'></ejs-pivotview></div>`
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    saveReport(args: any) {
      let reports = [];
      let isSaved: boolean = false;
      if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== '') {
        reports = JSON.parse(localStorage['pivotviewReports']);
      }
      if (args.report && args.reportName && args.reportName !== '') {
        reports.map(function (item: any): any {
          if (args.reportName === item.reportName) {
            item.report = args.report; isSaved = true;
          }
        });
        if (!isSaved) {
          reports.push(args);
        }
        localStorage['pivotviewReports'] = JSON.stringify(reports);
      }
    }
    fetchReport(args: any) {
      let reportCollection: string[] = [];
      let reeportList: string[] = [];
      if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== '') {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
      }
      reportCollection.map(function (item: any): void {
        reeportList.push(item.reportName);
      });
      args.reportName = reeportList;
    }
    loadReport(args: any) {
      let reportCollection: string[] = [];
      if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== '') {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
      }
    }
  }

```

```

    }
    reportCollection.map(function (item: any): void {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        this.pivotGridObj!.dataSourceSettings =
JSON.parse(args.report).dataSourceSettings;
    }
}
removeReport(args: any) {
    let reportCollection: any[] = [];
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
    }
    for (let i: number = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
    }
}
renameReport(args: any) {
    let reportCollection: string[] = [];
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
    }
    reportCollection.map(function (item: any): any { if (args.reportName
=== item.reportName) { item.reportName = args.rename; } });
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
    }
}
newReport() {
    this.pivotGridObj?.setProperties({ dataSourceSettings: { columns:
[], rows: [], values: [], filters: [] } }, false);
}
beforeExport(args: any): void {
    args.excelExportProperties = {
        header: {
            headerRows: 2,
            rows: [
                { cells: [{ colSpan: 4, value: "Pivot Table", style:
{ fontColor: '#C67878', fontSize: 20, hAlign: 'Center', bold: true,
underline: true } }] }
            ],
        },
        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 } }] }
        ]
    }
};
args.pdfExportProperties = {
    header: {
        fromTop: 0,
        height: 130,
        contents: [
            {
                type: 'Text',
                value: "Pivot Table",
                position: { x: 0, y: 50 },
                style: { textBrushColor: '#000000', fontSize:
13, dashStyle:'Solid',hAlign:'Center' }
            }
        ]
    },
    footer: {
        fromBottom: 160,
        height: 150,
        contents: [
            {
                type: 'PageNumber',
                pageNumberType: 'Arabic',
                format: 'Page {$current} of {$total}',
                position: { x: 0, y: 25 },
                style: { textBrushColor: '#02007a', fontSize: 15
}
            }
        ]
    }
};
}
ngOnInit(): void {
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'FieldList'] as ToolbarItems[];
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    }
}

```

```

    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionBegin

The event [actionBegin](#) triggers when the UI actions such as switching between pivot table and pivot chart, changing chart types, conditional formatting, exporting, etc. that are present in toolbar UI begin. This allows user to identify the current action being performed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action began. The following are the UI actions and their names:

Action	Action Name
-----	-----
New report	Add new report
Save report	Save current report
Save as report	Save as current report
Rename report	Rename current report
Remove report	Remove current report
Report change	Report change
Conditional Formatting	Open conditional formatting dialog
Number Formatting	Open number formatting dialog
Export menu	PDF export, Excel export, CSV export
Show Fieldlist	Open field list
Show Table	Show table view
Chart menu	Show chart view
Sub-totals menu	Hide sub-totals, Show row sub-totals, Show column sub-totals, Show sub-totals
Grand totals menu	Hide grand totals, Show row grand totals, Show column grand totals, Show grand totals

- **cancel**: It allows user to restrict the current action.

In the below sample, toolbar UI actions such as add new report and save current report can be restricted by setting the **args.cancel** option to **true** in the **actionBegin** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, FieldListService, CalculatedFieldService,
    NumberFormattingService,
    ToolbarService, ConditionalFormattingService, ToolbarItems,
    DisplayOption, IDataSet, PivotActionBeginEventArgs
} from 'syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [CalculatedFieldService, ToolbarService,
    ConditionalFormattingService, FieldListService, NumberFormattingService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings
allowExcelExport='true' allowNumberFormatting='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true'
allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' height='350'
(actionBegin)='actionBegin($event)' [toolbar]='toolbarOptions'></ejs-
pivotview></div>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    actionBegin(args: PivotActionBeginEventArgs): void {
        if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
            args.cancel = true;
        }
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
        'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'] as ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataSet[],
```

```

        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionComplete

The event [actionComplete](#) triggers when the UI actions such as switching between pivot table and pivot chart, changing chart types, conditional formatting, exporting, etc. that are present in toolbar UI, is completed. This allows user to identify the current UI actions being completed at runtime. It has the following parameters:

- **dataSourceSettings**: It holds the current data source settings such as input data source, rows, columns, values, filters, format settings and so on.
- **actionName**: It holds the name of the current action completed. The following are the UI actions and their names:

Action	Action Name
New report	New report added
Save report	Report saved
Save as report	Report re-saved
Rename report	Report renamed
Remove report	Report removed
Report change	Report changed
Conditional Formatting	Conditionally formatted
Number Formatting	Number formatted
Export menu	PDF exported, Excel exported, CSV exported
Show Fieldlist	Field list closed

| Show Table | Table view shown |

| Sub-totals menu | Sub-totals hidden, Row sub-totals shown, Column sub-totals shown, Sub-totals shown |

| Grand totals menu | Grand totals hidden, Row grand totals shown, Column grand totals shown, Grand totals shown |

- **actionInfo**: It holds the unique information about the current UI action. For example, while adding new report, the event argument contains information such as report name and the action name.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, FieldListService, CalculatedFieldService,
    NumberFormattingService,
    ToolbarService, ConditionalFormattingService, ToolbarItems,
    DisplayOption, IDataset, PivotActionCompleteEventArgs
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [CalculatedFieldService, ToolbarService,
    ConditionalFormattingService, FieldListService, NumberFormattingService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings
allowExcelExport='true' allowNumberFormatting='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true'
allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' (actionComplete)='actionComplete($event)'
height='350' [toolbar]='toolbarOptions'></ejs-pivotview></div>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    actionComplete(args: PivotActionCompleteEventArgs): void {
        if (args.actionName == 'New report added' || args.actionName ==
'Report saved') {
            // Triggers when the toolbar UI actions such as add new report
and save current report icon are completed.
        }
    }
}
```



```

    }
  }
  ngOnInit(): void {
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
    'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'] as ToolbarItems[];
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ActionFailure

The event [actionFailure](#) triggers when the current UI action fails to achieve the desired result. It has the following parameters:

- **actionName**: It holds the name of the current action failed. The following are the UI actions and their names:

Action	Action Name
-----	-----
New report	Add new report
Save report	Save current report
Save as report	Save as current report
Rename report	Rename current report
Remove report	Remove current report
Report change	Report change
Conditional Formatting	Open conditional formatting dialog

| Number Formatting | Open number formatting dialog |

| Export menu | PDF export, Excel export, CSV export |

| Show Fieldlist | Open field list |

| Show Table | Show table view |

| Chart menu | Show chart view |

| Sub-totals menu | Hide sub-totals, Show row sub-totals, Show column sub-totals, Show sub-totals |

| Grand totals menu | Hide grand totals, Show row grand totals, Show column grand totals, Show grand totals |

- **errorInfo**: It holds the error information of the current UI action.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
    IDataOptions, PivotView, FieldListService, CalculatedFieldService,
    NumberFormattingService,
    ToolbarService, ConditionalFormattingService, ToolbarItems,
    DisplayOption, IDataset, PivotActionFailureEventArgs
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [CalculatedFieldService, ToolbarService,
    ConditionalFormattingService, FieldListService, NumberFormattingService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings
allowExcelExport='true' allowNumberFormatting='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true'
allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' (actionFailure)='actionFailure($event)'
height='350' [toolbar]='toolbarOptions'></ejs-pivotview></div>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    actionFailure(args: PivotActionFailureEventArgs): void {
```

```

        if (args.actionName == 'Add new report' || args.actionName == 'Save
current report') {
            // Triggers when the current UI action fails to achieve the
desired result.
        }
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
        'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'NumberFormatting', 'FieldList'] as ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Toolbar Component](#)

Tool tip in Angular Pivotview component

The tooltip can be enabled or disabled by setting the [showTooltip](#) property to **true** or **false**. By default, tooltip is enabled in the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width
[showTooltip]='false'></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping:
false,
        minimumSignificantDigits: 1, maximumSignificantDigits: 3
}],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip Template

User can design their own tooltip by setting the property [tooltipTemplate](#) with own HTML elements. The property accepts both HTML string and ID attribute. The following place holders are available to display its dynamic values inside the HTML elements.

`${rowHeaders}` – Row headers of the selected value cell.

`${columnHeaders}` – Column headers of the selected value cell.

`${rowFields}` – Row fields of the selected value cell.

`${columnFields}` – Column fields of the selected value cell.

`${valueField}` – Field name of the selected value cell.

`${aggregateType}` – Aggregate type of the selected value cell.

`${value}` – Formatted value of the selected value cell.

The tooltip customization is common for both pivot table and pivot chart or it can be done individually as well. To customize the pivot table tooltip, the above procedure needs to be followed. To customize the pivot chart tooltip alone use `template` property of tooltip under [chartSettings](#).

In the below sample, the pivot table and pivot chart shows customized tooltip layouts.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, ToolbarService, ToolbarItems, DisplayOption,
  IDataset, PivotChartService
} from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ToolbarService, PivotChartService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings showToolbar='true'
width='100%' [displayOption]='displayOption' height='350'
tooltipTemplate='#Template' [toolbar]='toolbarOptions'
[chartSettings]='chartSettings'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public toolbarOptions?: ToolbarItems[];
  public displayOption?: DisplayOption;
  public chartSettings?: ChartSettings;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Column', animation: {
enable: false } },
      tooltip:{ template:'<span
class="wrap">${aggregateType} of ${valueField}: ${value}</span>' }
    } as ChartSettings;
    this.toolbarOptions = ['Grid', 'Chart'] as ToolbarItems[];
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
```

```

        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Css customization in Angular Pivotview component

Hiding Axis

The visibility of row, column, value and filter axis in Field List and Grouping Bar can be changed using custom CSS setting. To do so, please refer the code sample below:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotTable' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showGroupingBar='true'
showFieldList='true'></ejs-pivotview></div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],

```

```

        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Text Alignment

The alignment of text inside row headers, column headers, value cells and summary cells can be changed using custom CSS setting. To do so, please refer the code sample below:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showGroupingBar='true'
showFieldList='true'></ejs-pivotview></div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {

```

```

        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.width = "100%";
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize header, value and summary cell style

The elements in pivot table like header cell, value cell and summary cell style can be customized using built-in CSS names. To do so, please refer the code sample below:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width showGroupingBar='true'
showFieldList='true'></ejs-pivotview></div>`,
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;

```



```

public width?: string;
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Printing and Exporting

Print in Angular Pivotview component

The rendered pivot table can be printed directly from the browser by invoking the [print](#) method from the grid's instance. The below sample code illustrates the print option being invoked by an external button click.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  template: `<div class="col-md-2"><button ej-button
id='print'>Print</button></div>
<div class="col-md-8"><ejs-pivotview #pivotview id='PivotView'
height=height [dataSourceSettings]=dataSourceSettings width=width></ejs-
pivotview></div>`

```

```

}))
export class AppComponent implements OnInit {
  public width?: string;
  public height?: number;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.height = 350;
    this.dataSourceSettings = {
      expandAll: false,
      dataSource: Pivot_Data as IDataset[],
      columns: [{ name: 'Year' }, { name: 'Quarter' }],
      values: [{ name: 'Sold' }, { name: 'Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }]
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#print');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.grid.print();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Similarly, to print the pivot chart, use the [print](#) method from the chart's instance. The below sample code illustrates the print option being invoked by an external button click.

To use pivot chart, you need to inject the `PivotChart` module in the pivot table.

To display the pivot chart, set the [displayOption](#) property to either **Chart** or **Both**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions,
DisplayOption, PivotChartService } from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService, PivotChartService],
    template: `<div class="col-md-2"><button ej-button
id='print'>Print</button></div>
    <div class="col-md-8"><ejs-pivotview #pivotview id='PivotView'
height=height [dataSourceSettings]=dataSourceSettings
    [displayOption]='displayOption' [chartSettings]='chartSettings'
width=width></ejs-pivotview></div>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public height?: number;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public displayOption?: DisplayOption;
    public chartSettings?: ChartSettings;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
      this.width = "100%";
      this.height = 350;
      this.displayOption = { view: 'Chart' } as DisplayOption;
      this.chartSettings = { chartSeries: { type: 'Column' } } as
ChartSettings;
      this.dataSourceSettings = {
        expandAll: false,
        dataSource: Pivot_Data as IDataset[],
        columns: [{ name: 'Year' }, { name: 'Quarter' }],
        values: [{ name: 'Sold' }, { name: 'Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }]
      };
      this.button = new Button({ isPrimary: true });
      this.button.appendTo('#print');
      this.button.element.onclick = (): void => {
        this.pivotGridObj?.chart.print();
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Excel export in Angular Pivot Table component

The Excel export allows pivot table data to export as Excel document. To enable Excel export in the pivot table, set the `allowExcelExport` as **true**. You need to use the `excelExport` method for Excel exporting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowExcelExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.excelExport();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Multiple pivot table exporting

The Excel export provides an option to export multiple pivot table data in the same Excel file.

Same WorkSheet

The Excel export provides support to export multiple pivot tables in same sheet. To export in same sheet, define `multipleExport.type` as `AppendToSheet` in `ExcelExportProperties`. It has an option to provide blank rows between pivot tables and these blank row(s) count can be defined using the `multipleExport.blankRows` property.

By default, `multipleExport.blankRows` value is 5 between pivot tables within the same sheet.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview>
    <ejs-pivotview #pivotview1 id='PivotView1' height='350'
[dataSourceSettings]=dataSourceSettings1 allowExcelExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public dataSourceSettings1?: IDataOptions;
  public button?: Button;
  public excelExportProperties?: ExcelExportProperties;
  public firstGridExport?: Promise<any>;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  @ViewChild('pivotview1')
  public pivotGridObj1?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.dataSourceSettings1 = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.excelExportProperties = {
            multipleExport: { type: 'AppendToSheet', blankRows: 2 }
        };
        this.firstGridExport =
this.pivotGridObj?.grid.excelExport(this.excelExportProperties, true);
        this.firstGridExport?.then((fData: any) => {
            this.pivotGridObj1!.excelExport(this.excelExportProperties,
false, fData);
        });
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

New Worksheet

Excel export provides support to export multiple pivot tables into new sheets. To export in new sheets, define `multipleExport.type` as `NewSheet` in `ExcelExportProperties`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview>
    <ejs-pivotview #pivotview1 id='PivotView1' height='350'
[dataSourceSettings]=dataSourceSettings1 allowExcelExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public dataSourceSettings1?: IDataOptions;
  public button?: Button;
  public excelExportProperties?: ExcelExportProperties;
  public firstGridExport?: Promise<any>;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  @ViewChild('pivotview1')
  public pivotGridObj1?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.dataSourceSettings1 = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],

```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.excelExportProperties = {
            multipleExport: { type: 'NewSheet' }
        };
        this.firstGridExport =
this.pivotGridObj?.grid.excelExport(this.excelExportProperties, true);
        this.firstGridExport?.then((fData: any) => {
            (this.pivotGridObj1 as
PivotView).excelExport(this.excelExportProperties, false, fData);
        });
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing the pivot table style while exporting

The Excel export provides an option to change colors for headers, caption and records in pivot table before exporting. In-order to apply colors, define **theme** settings in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

By default, material theme is applied to exported Excel document.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

```



```

    ],
    standalone: true,
    selector: 'app-container',
    template: `<div class="col-md-8">
      <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview></div>
      <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public excelExportProperties?: ExcelExportProperties;
    public firstGridExport?: Promise<any>;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
      };
      this.width = '100%';
      this.button = new Button({ isPrimary: true });
      this.button.appendTo('#export');
      this.button.element.onclick = (): void => {
        this.excelExportProperties = {
          theme:
            {
              header: { fontName: 'Segoe UI', fontColor: '#666666' },
              record: { fontName: 'Segoe UI', fontColor: '#666666' },
              caption: { fontName: 'Segoe UI', fontColor: '#666666' }
            }
        };
        this.pivotGridObj?.excelExport(this.excelExportProperties);
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Add header and footer while exporting

The Excel export provides an option to include header and footer content for the excel document before exporting. In-order to add header and footer, define **header** and **footer** properties in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowExcelExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public excelExportProperties?: ExcelExportProperties;
  public firstGridExport?: Promise<any>;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
```

```

        this.excelExportProperties = {
            header: {
                headerRows: 2,
                rows: [
                    { cells: [{ colSpan: 4, value: "Pivot Table",
                        style: { fontColor: '#C67878', fontSize: 20, hAlign:
'Center', bold: true, underline: true } }] }
                ],
            },
            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.pivotGridObj?.excelExport(this.excelExportProperties);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing the file name while exporting

The Excel export provides an option to change file name of the document before exporting. In-order to change the file name, define **fileName** property in **excelExportProperties** object and pass it as a parameter to the [excelExport](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,

```

```

selector: 'app-container',
template: `<div class="col-md-8">
  <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview></div>
  <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
  ))
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public excelExportProperties?: ExcelExportProperties;
  public firstGridExport?: Promise<any>;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.excelExportProperties = {
        fileName: 'sample.xlsx'
      };
      this.pivotGridObj?.excelExport(this.excelExportProperties);
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Limitation when exporting millions of records to Excel 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 to the [documentation link](#) for more details on Microsoft Excel specifications and limits. Therefore, it is suggested to export the data in CSV (Comma-Separated Values) or other formats that can handle large datasets more efficiently than Excel.

CSV Export

Also, the Excel export allows pivot table data to be exported in **CSV** file format. To export pivot table in **CSV** file format, you need to use the **csvExport** method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public excelExportProperties?: ExcelExportProperties;
    public firstGridExport?: Promise<any>;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
        };
        this.width = '100%';
        this.button = new Button({ isPrimary: true });
        this.button.appendTo('#export');
        this.button.element.onclick = (): void => {
            this.pivotGridObj?.csvExport();
        }
    }
}
```

```

    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Virtual Scroll Data

You can export the pivot table virtual scroll data as Excel/CSV document by using PivotEngine export without any performance degradation. To enable PivotEngine export in the pivot table, set the `allowExcelExport` as true. You need to use the `exportToExcel` method for PivotEngine export.

To use PivotEngine export, You need to inject the `ExcelExport` module in pivot table.

PivotEngine export will be performed while enabling virtual scrolling by default.

Virtual Scroll Data Excel Export

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, ExcelExportService } from
'@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ExcelExportService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {

```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.excelExportModule.exportToExcel('Excel');
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Virtual Scroll Data CSV Export

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, ExcelExportService } from
'@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [ExcelExportService],
  template: `<div class="col-md-8">
    <ejs-pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})

```

```

export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      let excelExportProperties = {
        fileName: 'csvexport.csv',
      };
      this.pivotGridObj?.csvExport(excelExportProperties);
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export all pages

The pivot engine exports the entire virtual data of the pivot table (i.e. the data that contains all of the records used to render the complete pivot table) as an Excel/CSV document. To export just the current viewport of the pivot table, set the [exportAllPages](#) property to **false**. To use the pivot engine export, add the **ExcelExport** module into the pivot table.

By default, the pivot engine export will be performed while virtual scrolling is enabled.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';

```



```

import { IDataOptions, IDataset, PivotView, VirtualScrollService } from
'@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [VirtualScrollService],
template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowExcelExport='true'
exportAllPages='false' enableVirtualization='true' width=width></ejs-
pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            values: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.width = '100%';
        this.button = new Button({ isPrimary: true });
        this.button.appendTo('#export');
        this.button.element.onclick = (): void => {
            this.pivotGridObj?.excelExport();
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Events

ExcelQueryCellInfo

The event `excelQueryCellInfo` triggers while framing each row and value cell during Excel export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `value` - It holds the cell value.
- `column` - It holds column information for the current cell.
- `data` - It holds the entire row data across the current cell.
- `style` - It holds the style properties for the cell.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]='dataSourceSettings'
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public columnGrandTotalIndex?: number;
  public rowGrandTotalIndex?: number;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotView;
  excelQueryCellInfo(args: any): void {
    ((this.pivotGridObj as PivotView).renderModule as
any).columnCellBoundEvent(args);
    //triggers for every cell while exporting
  }
  enginePopulated(args: any): void {
    ((this.pivotGridObj as PivotView).grid as Grid).excelQueryCellInfo =
this.excelQueryCellInfo.bind(this);
  }
}
```

```

ngOnInit(): void {
  this.width = '100%';
  this.dataSourceSettings = {
    dataSource: Pivot_Data as IDataset[],
    expandAll: false,
    drilledMembers: [{ name: 'Country', items: ['France'] }],
    columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
    values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
    rows: [{ name: 'Country' }, { name: 'Products' }],
    formatSettings: [{ name: 'Amount', format: 'C0' }],
    filters: []
  };
  this.gridSettings = {
    columnWidth: 140,
  } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ExcelHeaderQueryCellInfo

The event `excelHeaderQueryCellInfo` triggers on framing each header cell during Excel export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `cell` - It holds the current cell information.
- `style` - It holds the style properties for the cell.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDatasetOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,

```

```

selector: 'app-container',
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
}))
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public columnGrandTotalIndex?: number;
  public rowGrandTotalIndex?: number;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotView;
  excelHeaderQueryCellInfo(args: any): void {
    ((this.pivotGridObj as PivotView).renderModule as
any).columnCellBoundEvent(args);
    //triggers for every header cell while excel exporting
  }
  enginePopulated(args: any): void {
    ((this.pivotGridObj as PivotView).grid as
Grid).excelHeaderQueryCellInfo = this.excelHeaderQueryCellInfo.bind(this);
  }
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.gridSettings = {
      columnWidth: 140,
    } as GridSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ExportComplete

The event [exportComplete](#) is triggered after the pivot table data has been exported to an Excel/CSV document. You can use this event to acquire blob stream data for further customization and processing

at your end by passing the `isBlob` parameter as `true` when using the [excelExport](#) method. It has the following parameters:

- `type` - It holds the current export type such as PDF, Excel, and CSV.
- `promise` - It holds the promise object for blob data.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService,
 ExportCompleteEventArgs, ExcelExportService } from '@syncfusion/ej2-angular-
 pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { ExcelExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService, ExcelExportService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowExcelExport='true'
    enableVirtualization='true' (exportComplete)=exportComplete($event) '
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  exportComplete(args: ExportCompleteEventArgs | any): void {
    args.promise.then((e: { blobData: Blob }) => {
      console.log(e.blobData);
    });
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
      rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
 'Amount', caption: 'Sold Amount' }],
      values: [{ name: 'Country' }, { name: 'Products' }],
```

```

        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        let excelExportProperties: ExcelExportProperties = { };
        this.pivotGridObj?.excelExport(excelExportProperties, false,
null, true);
    };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [PDF Exporting](#)

Pdf export in Angular Pivotview component

PDF export allows exporting pivot table data as PDF document. To enable PDF export in the pivot table, set the `allowPdfExport` as true. You need to use the `pdfExport` method for PDF exporting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})

```

```

export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.pdfExport();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Multiple pivot table exporting

PDF export provides an option for exporting multiple pivot tables to same file. In this exported document, each pivot table will be exported to new page of document in same file.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,

```

```

        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    template: `<div class="col-md-8">
        <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview>
        <ejs-pivotview #pivotview1 id='PivotView1'
[dataSourceSettings]=dataSourceSettings1 allowPdfExport='true'
width=width></ejs-pivotview></div>
        <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
    })
    export class AppComponent implements OnInit {
        public width?: string;
        public dataSourceSettings?: IDataOptions;
        public dataSourceSettings1?: IDataOptions;
        public button?: Button;
        public firstGridPdfExport?: Promise<Object>;
        @ViewChild('pivotview', {static: false})
        public pivotGridObj?: PivotView;
        @ViewChild('pivotview1')
        public pivotGridObj1?: PivotView;
        ngOnInit(): void {
            this.width = "100%";
            this.dataSourceSettings = {
                dataSource: Pivot_Data as IDataset[],
                expandAll: false,
                columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
                values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
                rows: [{ name: 'Country' }, { name: 'Products' }],
                formatSettings: [{ name: 'Amount', format: 'C0' }],
                filters: [],
                valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
            };
            this.dataSourceSettings1 = {
                dataSource: Pivot_Data as IDataset[],
                expandAll: false,
                columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
                values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
                rows: [{ name: 'Country' }, { name: 'Products' }],
                formatSettings: [{ name: 'Amount', format: 'C0' }],
                filters: [],
                valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
            };
            this.button = new Button({ isPrimary: true });
            this.button.appendTo('#export');
            this.button.element.onclick = (): void => {
                this.firstGridPdfExport = this.pivotGridObj?.grid.pdfExport({},
true);
                this.firstGridPdfExport?.then((pdfData: Object) => {

```



```

        this.pivotGridObj1?.pdfExport({}, false, pdfData);
    });
};
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export table and chart into the same document

When the [displayOption](#) is set to **Both**, you can export both the table and the chart into the same PDF document. To achieve this, use the [pdfExport](#) method and set the `exportBothTableAndChart` parameter to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, PDFExportService, DisplayOption
} from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PDFExportService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
[displayOption]='displayOption' [chartSettings]='chartSettings'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public displayOption?: DisplayOption;
  public chartSettings?: ChartSettings;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;

```

```

ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
    this.displayOption = { view: 'Both' } as DisplayOption;
    this.chartSettings = { chartSeries: { type: 'Column' } } as
ChartSettings;
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.pdfExport(undefined, false, undefined, false,
true);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customization during PDF export

PDF export provides option to customize mapping of pivot table to the exported PDF document.

To add header and footer while exporting

You can customize text, page number, line, page size and changing orientation in header and footer of the exported document.

To add a text in header/footer

You can add text either in header or footer of the exported PDF document like in the below code example.

`typescript

```

let pdfExportProperties: PdfExportProperties = {
    header: {
        fromTop: 0,
        height: 130,
        contents: [
            {

```

```

type: 'Text',
value: "Northwind Traders",
position: { x: 0, y: 50 },
style: { textBrushColor: '#000000', fontSize: 13 }
},
]
}
}
,

```

To draw a line in header/footer

You can add line either in header or footer of the exported PDF document like in the below code example.

Supported line styles:

- dash
- dot
- dashdot
- dashdotdot
- solid

```

`typescript
let pdfExportProperties: 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/footer

You can add page number either in header or footer of exported PDF document like in the below code example.

Supported page number types:

- LowerLatin - a, b, c,
- UpperLatin - A, B, C,
- LowerRoman - i, ii, iii,
- UpperRoman - I, II, III,
- Number - 1,2,3.

```
`typescript
let pdfExportProperties: 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' }
}
]
}
}
```

The below code illustrates the PDF export customization options.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from '../datasource';
```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public pdfExportProperties?: PdfExportProperties;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pdfExportProperties = {
        header: {
          fromTop: 0,
          height: 130,
          contents: [
            {
              type: 'Text',
              value: "Pivot Table",
              position: { x: 0, y: 50 },
              style: { textBrushColor: '#000000', fontSize:
13, dashStyle: 'Solid', hAlign: 'Center' }
            }
          ]
        },
        footer: {
          fromBottom: 160,
          height: 150,
          contents: [

```

```

        {
            type: 'PageNumber',
            pageNumberType: 'Arabic',
            format: 'Page {$current} of {$total}',
            position: { x: 0, y: 25 },
            style: { textBrushColor: '#02007a', fontSize: 15 }
        }
    ]
}
};
this.pivotGridObj?.pdfExport(this.pdfExportProperties);
};
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Add an image in header/footer

You can add image (Base64 string) either in header or footer of the exported PDF document like in the below code example.

`typescript

```
let pdfExportProperties: PdfExportProperties = {
```

```
  header: {
```

```
    fromTop: 0,
```

```
    height: 130,
```

```
    contents: [
```

```
    {
```

```
      type: 'Image',
```

```
      src: image,
```

```
      position: { x: 20, y: 10 },
```

```
      size: { height: 100, width: 100 },
```

```
    }
```

```
  ]
```

```
}
```

```
}
```

```
,
```

The below code illustrates the PDF export customization options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { image } from './image';
import { Pivot_Data } from './datasource';
@Component({
imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
standalone: true,
    selector: 'app-container',
    template: `<div class="col-md-8">
        <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview></div>
        <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public pdfExportProperties?: PdfExportProperties;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
        this.width = "100%";
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
        };
        this.button = new Button({ isPrimary: true });
        this.button.appendTo('#export');
        this.button.element.onclick = (): void => {
            this.pdfExportProperties = {
                header: {
                    fromTop: 0,
                    height: 130,
                    contents: [

```

```

        {
            type: 'Image',
            src: image,
            position: { x: 20, y: 10 },
            size: { height: 100, width: 100 },
        }
    ]
}
};
this.pivotGridObj?.pdfExport(this.pdfExportProperties);
};
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing the file name while exporting

The PDF export provides an option to change file name of the document before exporting. In-order to change the file name, define **fileName** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;

```



```

public button?: Button;
public pdfExportProperties?: PdfExportProperties;
@ViewChild('pivotview', {static: false})
public pivotGridObj?: PivotView;
ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pdfExportProperties = {
            fileName: 'sample.pdf'
        };
        this.pivotGridObj?.pdfExport(this.pdfExportProperties);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing page orientation while exporting

The PDF export provides an option to change page orientation of the document before exporting. In-order to change the page orientation, define **pageOrientation** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method. By default, the page orientation will be in **Portrait** and it can be changed to **Landscape** based on user requirement.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';

```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public pdfExportProperties?: PdfExportProperties;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pdfExportProperties = {
        pageOrientation: 'Landscape'
      };
      this.pivotGridObj?.pdfExport(this.pdfExportProperties);
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing page size while exporting

The PDF export provides an option to change page size of the document before exporting. In-order to change the page size, define **pageSize** property in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

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.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public pdfExportProperties?: PdfExportProperties;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
```

```

    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pdfExportProperties = {
            pageSize: 'Letter'
        };
        this.pivotGridObj?.pdfExport(this.pdfExportProperties);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing document width and height while exporting

Before exporting, you can change the height and width of the PDF document. To achieve this, use the **height** and **width** properties in the [beforeExport](#) event.

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the **VirtualScroll** and **PDFExport** modules must be injected into the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview';
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService, PDFExportService, BeforeExportEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PDFExportService, VirtualScrollService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    enableVirtualization='true' (beforeExport)='beforeExport($event)'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {

```

```

public width?: string;
public dataSourceSettings?: IDataOptions;
public button?: Button;
@ViewChild('pivotview', {static: false})
public pivotGridObj?: PivotView;
beforeExport(args: BeforeExportEventArgs) {
    args.width = this.pivotGridObj?.element.offsetWidth;
    args.height = this.pivotGridObj?.element.offsetHeight;
}
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.pdfExport();
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize the table column count while exporting

Before exporting, you can split and export the pivot table columns on each page of the PDF document by using the **columnSize** property in the [beforeExport](#) event.

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the **VirtualScroll** and **PDFExport** modules must be injected into the pivot table.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, VirtualScrollService, PDFExportService,
BeforeExportEventArgs, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';

```

```

@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  providers: [PDFExportService,VirtualScrollService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    enableVirtualization='true' (beforeExport)='beforeExport($event)'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  beforeExport(args: BeforeExportEventArgs) {
    args.columnSize = 6;
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      values: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.pdfExport();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing the table's column width and row height while exporting

You can change the column width and row height in the PDF document during the pivot table export by using the [onPdfCellRender](#) event. Within this event, the `args.column.width` property allows you to change the width of specific columns.

As shown in the code example below, the “Unit Sold” column under “FY 2015” is changed to a width of 60 pixels.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, VirtualScrollService, PDFExportService,
  IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [PDFExportService, VirtualScrollService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    enableVirtualization='true' (onPdfCellRender)='onPdfCellRender($event)'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  onPdfCellRender(args: any) {
    if (args.pivotCell && args.pivotCell.valueSort &&
    args.pivotCell.valueSort.levelName === 'FY 2015.Units Sold') {
      args.column.width = 60
    }
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
      'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
      'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
    }
  }
}
```

```

        filters: [],
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.pdfExport({}, false, undefined, false, true);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Similarly, you can change the height of specific rows in the PDF document by using the `args.cell.height` property in the [onPdfCellRender](#) event.

As shown in the code example below, the “Mountain Bikes” row under “France” is changed to a height of 30 pixels.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, PivotView, VirtualScrollService, PDFExportService,
IDataSet } from '@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
imports: [

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
standalone: true,
selector: 'app-container',
providers: [PDFExportService, VirtualScrollService],
template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
enableVirtualization='true' (onPdfCellRender)='onPdfCellRender($event)'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    @ViewChild('pivotview', {static: false})

```



```

public pivotGridObj?: PivotView;
onPdfCellRender(args: any) {
    if (args.pivotCell && args.pivotCell.valueSort &&
args.pivotCell.valueSort.levelName === 'France.Mountain Bikes') {
        args.cell.height = 30
    }
}
ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.pdfExport({}, false, undefined, false, true);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

This option is only available if [enableVirtualization](#) is set to **true**. In addition, the **VirtualScroll** and **PDFExport** modules must be injected into the pivot table.

Changing the pivot table style while exporting

The PDF export provides an option to change colors for headers, caption and records in pivot table before exporting. In-order to apply colors, define **theme** settings in **pdfExportProperties** object and pass it as a parameter to the [pdfExport](#) method.

By default, material theme is applied to exported PDF document.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDatasetOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';

```

```

import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

],
standalone: true,
selector: 'app-container',
template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public button?: Button;
    public pdfExportProperties?: PdfExportProperties;
    @ViewChild('pivotview', {static: false})
    public pivotGridObj?: PivotView;
    ngOnInit(): void {
        this.width = "100%";
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: [],
            valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
        };
        this.button = new Button({ isPrimary: true });
        this.button.appendTo('#export');
        this.button.element.onclick = (): void => {
            this.pdfExportProperties = {
                theme: {
                    header: {
                        17, bold: true,
                        fontColor: '#64FA50', fontName: 'Calibri', fontSize:
                        borders: { color: '#64FA50', lineStyle: 'Thin' }
                    },
                    record: {
                        17, bold: true
                        fontColor: '#64FA50', fontName: 'Calibri', fontSize:
                        caption: {
                        17, bold: true
                        fontColor: '#64FA50', fontName: 'Calibri', fontSize:
                    }
                }
            }
        }
    }
}

```

```

    }
    } as PdfExportProperties;
    this.pivotGridObj?.pdfExport(this.pdfExportProperties);
  };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD009 -->

Changing default font while exporting

By default, the pivot table uses "Helvetica" font in the exported document. But it can be changed using the [theme](#) property in [pdfExportProperties](#).

The available built-in fonts are,

- Helvetica
- TimesRoman
- Courier
- Symbol
- ZapfDingbats

`typescript

```
import { PdfStandardFont, PdfFontFamily, PdfFontStyle } from '@syncfusion/ej2-pdf-export';
```

...

```
let pdfExportProperties: 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) }
```

```
  }
```

```
};
```

```
,
```

Adding custom font while exporting

In addition to existing built-in fonts, custom fonts can also be used. The custom font should be in **Base64** format and mention it in **PdfTrueTypeFont** class. In the following example, we have used **Advent Pro** font family that supports **Hungarian** language.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data, base64AlgeriaFont } from './datasource';
import { PdfTrueTypeFont } from '@syncfusion/ej2-pdf-export';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public pdfExportProperties?: PdfExportProperties;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      columns: [{ name: 'Year', caption: 'Production Year' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pdfExportProperties = {
        theme: {
          header: {font: new PdfTrueTypeFont(base64AlgeriaFont, 11)
},
          caption: { font: new PdfTrueTypeFont(base64AlgeriaFont, 9)
},
          record: { font: new PdfTrueTypeFont(base64AlgeriaFont, 10) }
        }
      };
      this.pivotGridObj?.pdfExport(this.pdfExportProperties);
    };
  };
}

```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The non-English alphabets can also be exported properly by setting its appropriate font.

Virtual Scroll Data

You can export the pivot table virtual scroll data as PDF document by using PivotEngine export without any performance degradation. To enable PivotEngine export in the pivot table, set the `allowPdfExport` as true. You need to use the `exportToPDF` method for PivotEngine export.

To use PivotEngine export, You need to inject the `PDFExport` module in pivot table.

PivotEngine export will be performed while enabling virtual scrolling by default

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
    [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
    width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
```

```

        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        this.pivotGridObj?.pdfExportModule.exportToPDF();
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Repeat row headers

Repeat row headers on each page can be achieved using PivotEngine export option. To disable repeat row headers, you need to set `allowRepeatHeader` to `false` in `beforeExport` event. You need to use the `exportToPDF` method for PivotEngine export.

To use PivotEngine export, You need to inject the `PDFExport` module in pivot table.

By default, repeat row headers is enabled in the PivotEngine export.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="col-md-8">

```

```

<ejs-pivotview #pivotview id='PivotView' height='350'
(beforeExport)='beforeExport($event)'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
width=width></ejs-pivotview></div>
<div class="col-md-2"><button ej-button id='export'>Export</button></div>`
}))
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  beforeExport(args: any) {
    args.allowRepeatHeader = false;
  }
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: [],
      valueSortSettings: { headerText: 'FY 2015##Q1##Amount',
headerDelimiter: '##', sortOrder: 'Descending' }
    };
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.pdfExportModule.exportToPDF();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export all pages

The pivot engine exports the entire virtual data of the pivot table (i.e. the data that contains all of the records used to render the complete pivot table) as a PDF document. To export just the current viewport of the pivot table, set the [exportAllPages](#) property to **false**. To use the pivot engine export, add the **PDFExport** module into the pivot table.

By default, the pivot engine export will be performed while virtual scrolling is enabled.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService } from
'@syncfusion/ej2-angular-pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
exportAllPages='false' enableVirtualization='true' width=width></ejs-
pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      values: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
      this.pivotGridObj?.pdfExport();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```



```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Events

PdfQueryCellInfo

The event `pdfQueryCellInfo` triggers on framing each row and value cell during PDF export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `value` - It holds the cell value.
- `column` - It holds column information for the current cell.
- `data` - It holds the entire row data across the current cell.
- `style` - It holds the style properties for the cell.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public gridSettings?: GridSettings;
  public columnGrandTotalIndex: any;
  public rowGrandTotalIndex: any;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: PivotView;
  pdfQueryCellInfo(args: any): void {
    ((this.pivotGridObj as PivotView).renderModule as
any).columnCellBoundEvent(args);
    //triggers for every cell while exporting
  }
}
```

```

enginePopulated(args: any): void {
    ((this.pivotGridObj as PivotView).grid as Grid).pdfQueryCellInfo =
    this.pdfQueryCellInfo.bind(this);
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.gridSettings = {
        columnWidth: 140,
    } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

PdfHeaderQueryCellInfo

The event `pdfHeaderQueryCellInfo` triggers on framing each column header cell during PDF export. It allows the user to customize the cell value, style etc. of the current cell. It has the following parameters:

- `cell` - It holds the current rendering cell information.
- `style` - It holds the style properties for the cell.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-angular-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' (enginePopulated)='enginePopulated($event)'
width=width></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    public columnGrandTotalIndex?: number;
    public rowGrandTotalIndex?: number;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    pdfHeaderQueryCellInfo(args: any): void {
      ((this.pivotGridObj as PivotView).renderModule as
any).columnCellBoundEvent(args);
      //triggers for every header cell while exporting
    }
    enginePopulated(args: any): void {
      ((this.pivotGridObj as PivotView).grid as
Grid).pdfHeaderQueryCellInfo = this.pdfHeaderQueryCellInfo.bind(this);
    }
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.gridSettings = {
        columnWidth: 140,
      } as GridSettings;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

ExportComplete

The event [exportComplete](#) is triggered after the pivot table data has been exported to a PDF document. You can use this event to acquire blob stream data for further customization and processing at your end by passing the `isBlob` parameter as `true` when using the [pdfExport](#) method. It has the following parameters:

- `type` - It holds the current export type such as PDF, Excel, and CSV.
- `promise` - It holds the promise object for blob data.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, VirtualScrollService,
 ExportCompleteEventArgs, PDFExportService } from '@syncfusion/ej2-angular-
 pivotview';
import { Button } from '@syncfusion/ej2-buttons';
import { PdfExportProperties } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [VirtualScrollService, PDFExportService],
  template: `<div class="col-md-8">
    <ejs-pivotview #pivotview id='PivotView' height='350'
 [dataSourceSettings]=dataSourceSettings allowPdfExport='true'
 enableVirtualization='true' (exportComplete)= 'exportComplete($event)'
 width=width></ejs-pivotview></div>
    <div class="col-md-2"><button ej-button id='export'>Export</button></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  exportComplete(args: ExportCompleteEventArgs | any): void {
    args.promise.then((e: { blobData: Blob }) => {
      console.log(e.blobData);
    });
  }
  ngOnInit(): void {
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: true,
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
 'Quarter' }],
```

```

        rows: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        values: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
    this.button = new Button({ isPrimary: true });
    this.button.appendTo('#export');
    this.button.element.onclick = (): void => {
        let pdfExportProperties: PdfExportProperties = { };
        this.pivotGridObj?.pdfExport(pdfExportProperties, false,
undefined, true);
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Excel Exporting](#)

Globalization and localization in Angular Pivotview 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](#)) & adding culture specific customization and translation to the text ([Localization](#)).

Internationalization

Internationalization library provides support for formatting and parsing the number, date, and time by using the official [Unicode CLDR](#) JSON data and also provides the `loadCldr` method to load the culture specific CLDR JSON data.

By default, 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 (it installs the CLDR JSON data). For more information about CLDR-Data, refer to this [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`.

- Download the required locale packages to render the angular Pivot Table component with specified locale. To download the locale definition of angular components, use this [link](#).
- Now import the required culture from the installed location to `app.ts` file, like the below code snippets.

```
`typescript
//import the loadCldr from ej2-base
import { loadCldr } from '@syncfusion/ej2-base';

loadCldr(
  require('cldr-data/supplemental/numberingSystems.json'),
  require('cldr-data/main/fr-CH/ca-gregorian.json'),
  require('cldr-data/main/fr-CH/numbers.json'),
  require('cldr-data/main/fr-CH/timeZoneNames.json'));
`
```

The Internationalization library is used to globalize number, date, and time values in pivot table component using the `dataSourceSettings.formatSettings` option.

- Set the culture by using the `locale` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { L10n, setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
import { IDataOptions, IDataset, FieldListService, CalculatedFieldService,
 GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from '../datasource';
setCulture('de');
setCurrencyCode('EUR');
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, CalculatedFieldService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width
  allowCalculatedField='true' showGroupingBar='true' locale='de-DE'
showFieldList='true'></ejs-pivotview></div>`
})
```

```

export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
        L10n.load({
            'de-DE': {
                'pivotview': {
                    'grandTotal': 'Gesamtsumme',
                    'total': 'Insgesamt',
                    'value': 'Wert',
                    'noValue': 'Kein Wert',
                    'row': 'Zeile',
                    'column': 'Spalte',
                    'collapse': 'Zusammenbruch',
                    'expand': 'Erweitern'
                },
                "pivotfieldlist": {
                    'fieldList': 'Feld Liste',
                    'dropRowPrompt': 'Drop Reihe hier',
                    'dropColPrompt': 'Drop column Hier',
                    'dropValPrompt': 'Drop wert hier',
                    'dropFilterPrompt': 'Drop Filter Hier',
                    'addPrompt': 'Feld hinzufügen',
                    'centerHeader': 'Ziehen Sie die Felder zwischen den
Bereichen unten:',
                    'add': 'Hinzufügen',
                    'drag': 'Ziehen',
                    'filters': 'Filter',
                    'rows': 'Zeilen',
                    'columns': 'Spalten',
                    'values': 'Werte',
                    'error': 'Fehler',
                    'dropAction': 'Berechnetes Feld nicht in jeder anderen
Region außer Wert Achse sein.',
                    'search': 'Suche',
                    'close': 'Schließen',
                    'cancel': 'Abbrechen',
                    'delete': 'Löschen',
                    'alert': 'Warnung',
                    'warning': 'Warnung',
                    'ok': 'OK',
                    'allFields': 'Alle Felder',
                    'noMatches': 'Keine Treffer'
                }
            }
        });
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C2', useGrouping:
false,

```

```

        minimumSignificantDigits: 1, maximumSignificantDigits: 3,
        currency: 'EUR' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
        '"Sum(Amount)"+"Sum(Sold) "' }]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

* In the above sample, **Amount** field is formatted by [NumberFormatOptions](#). For date formats, the value strings are formatted by [DateFormatOptions](#).

* By default, **locale** value is **en-US**. If you want to change the **en-US** culture to a different culture, you have to change the **locale** accordingly.

* Also, you will find more details about support format string for number formats and data formats [here](#).

Decimal separators

<!-- markdownlint-disable MD009 -->

The decimal separators of pivot table values varies based on the culture applied to the component. The culture can be set by calling the method [setCulture](#) with appropriate culture string as its parameter.

The following example demonstrates the decimal separators in **Deutsch** culture.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { loadCldr, L10n, setCulture, setCurrencyCode } from
'@syncfusion/ej2-base';
import { IDataOptions, IDataset, FieldListService, CalculatedFieldService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
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';
loadCldr(currencies, cagregorian, numbers, timeZoneNames, numberingSystems);
setCulture('de');
setCurrencyCode('EUR');
@Component({
  imports: [

```



```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService, CalculatedFieldService, GroupingBarService],
    // specifies the template string for the pivot table component
    template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width
    allowCalculatedField='true' showGroupingBar='true' locale='de-DE'
showFieldList='true'></ejs-pivotview></div>`
  })
  export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {

      L10n.load({
        'de-DE': {
          'pivotview': {
            'grandTotal': 'Gesamtsumme',
            'total': 'Insgesamt',
            'value': 'Wert',
            'noValue': 'Kein Wert',
            'row': 'Zeile',
            'column': 'Spalte',
            'collapse': 'Zusammenbruch',
            'expand': 'Erweitern'
          },
          "pivotfieldlist": {
            'fieldList': 'Feld Liste',
            'dropRowPrompt': 'Drop Reihe hier',
            'dropColPrompt': 'Drop column Hier',
            'dropValPrompt': 'Drop wert hier',
            'dropFilterPrompt': 'Drop Filter Hier',
            'addPrompt': 'Feld hinzufügen',
            'centerHeader': 'Ziehen Sie die Felder zwischen den
Bereichen unten:',
            'add': 'Hinzufügen',
            'drag': 'Ziehen',
            'filters': 'Filter',
            'rows': 'Zeilen',
            'columns': 'Spalten',
            'values': 'Werte',
            'error': 'Fehler',
            'dropAction': 'Berechnetes Feld nicht in jeder anderen
Region außer Wert Achse sein.',
            'search': 'Suche',
            'close': 'Schließen',
            'cancel': 'Abbrechen',
            'delete': 'Löschen',
            'alert': 'Warnung',
            'warning': 'Warnung',
            'ok': 'OK',
            'allFields': 'Alle Felder',

```

```

        'noMatches': 'Keine Treffer'
    }
    });
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C2', currency: 'EUR'
}],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold) "' }]]];
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Localization

The [Localization](#) library allows you to localize default text content of the pivot table. The pivot table component has static text on some features (like drop area text, pivot field list title, 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 pivot table.

Locale keywords | Text

staticFieldList | Pivot Field List

fieldList | Field List

dropFilterPrompt | Drop filter here

dropColPrompt | Drop column here

dropRowPrompt | Drop row here

dropValPrompt | Drop value here

addPrompt | Add field here

adaptiveFieldHeader | Choose field

centerHeader | Drag fields between axes below:

add | add

drag | Drag

filter | Filter

filtered | Filtered

sort | Sort

remove | Remove

filters | Filters

rows | Rows

columns | Columns

values | Values

CalculatedField | Calculated Field

createCalculatedField | Create Calculated Field

fieldName | Enter the field name

error | Error

invalidFormula | Invalid formula.

dropText | Example: ('Sum(*OrderCount*)' + 'Sum(*InStock*)') * 250

dropTextMobile | Add fields and edit formula here.

dropAction | Calculated field cannot be place in any other region except value axis.

search | Search

close | Close

cancel | Cancel

delete | Delete

alert | Alert

warning | Warning

ok | OK

Sum | Sum

Avg | Avg

Count | Count

Min | Min

Max | Max

allFields | All Fields

formula | Formula

fieldExist | A field already exists in this name. Please enter a different name.

confirmText | A calculation field already exists in this name. Do you want to replace it?

noMatches | No matches

format | Summaries values by

edit | Edit

clear | Clear

formulaField | Drag and drop fields to formula

dragField | Drag field to formula

clearFilter | Clear

by | by

enterValue | Enter value

chooseDate | Enter date

all | All

multipleItems | Multiple items

Equals | Equals

DoesNotEquals | Does Not Equal

BeginWith | Begins With

DoesNotBeginWith | Does Not Begin With

EndsWith | Ends With

DoesNotEndsWith | Does Not End With

Contains | Contains

DoesNotContains | Does Not Contain

GreaterThan | Greater Than

GreaterThanOrEqualTo | Greater Than Or Equal To

LessThan | Less Than

LessThanOrEqualTo | Less Than Or Equal To

Between | Between

NotBetween | Not Between

Before | Before

BeforeOrEqualTo | Before Or Equal To

After | After

AfterOrEqualTo | After Or Equal To

member | Member

label | Label

date | Date

value | Value

labelTextContent | Show the items for which the label

dateTextContent | Show the items for which the date

valueTextContent | Show the items for which

And | and

DistinctCount | Distinct Count

Product | Product

Index | Index

SampleStDev | Sample StDev

PopulationStDev | Population StDev

SampleVar | Sample Var

PopulationVar | Population Var

RunningTotals | Running Totals

DifferenceFrom | Difference From

PercentageOfDifferenceFrom | % of Difference From

PercentageOfGrandTotal | % of Grand Total

PercentageOfColumnTotal | % of Column Total

PercentageOfRowTotal | % of Row Total

PercentageOfParentTotal | % of Parent Total

PercentageOfParentColumnTotal | % of Parent Column Total

PercentageOfParentRowTotal | % of Parent Row Total

Years | Years

Quarters | Quarters

Months | Months

Days | Days

Hours | Hours

Minutes | Minutes

Seconds | Seconds

apply | APPLY

valueFieldSettings | Value field settings

sourceName | Field name :

sourceCaption | Field caption :

summarizeValuesBy | Summarize values by :

baseField | Base field :

baseItem | Base item :

example | e.g:

editorDataLimitMsg | more items. Search to refine further.

deferLayoutUpdate | Defer Layout Update

null | null

undefined | undefined

groupOutOfRange | Out of Range

fieldDropErrorAction | The field you are moving cannot be placed in that area of the report

MoreOption | More...

memberType | Field Type

selectedHierarchy | Parent Hierarchy

formatString | Format String

expressionField | Expression

olapDropText | Example: [Measures].[Order Quantity] + ([Measures].[Order Quantity] * 0.10)

customFormat | Enter custom format string

Measure | Measure

Dimension | Dimension

Standard | Standard

Currency | Currency

Percent | Percent

Custom | Custom

blank | (Blank)

fieldTooltip | Drag and drop fields to create an expression. And, if you want to edit the existing the calculated fields! You can achieve it by simply selecting the field under 'Calculated Members'.

fieldTitle | Field Name

QuarterYear | Quarter Year

caption | Field Caption

copy | Copy

group | Group

numberFormatString | Example: C, P, 0000 %, ###0.##0#, etc.

sortAscending | Sort ascending order

sortDescending | Sort descending order

sortNone | Sort data order

clearCalculatedField | Clear edited field info

editCalculatedField | Edit calculated field

selectGroup | Select groups

of | of

removeCalculatedField | Are you sure you want to delete this calculated field?

yes | Yes

no | No

None | None

Note: To find the latest localization keywords of pivotview and pivotfieldlist for different languages, visit this [GitHub](#) repository.

Loading Translations

To load translation object in an application, use [load](#) function of the [L10n](#) class.

The following example demonstrates the pivot table in **Deutsch** culture.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
import { IDataOptions, IDataset, FieldListService, CalculatedFieldService,
 GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, CalculatedFieldService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
[width]=width
  allowCalculatedField='true' showGroupingBar='true' locale='de-DE'
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    L10n.load({
      'de-DE': {
        'pivotview': {
```

```

        'grandTotal': 'Gesamtsumme',
        'total': 'Insgesamt',
        'value': 'Wert',
        'noValue': 'Kein Wert',
        'row': 'Zeile',
        'column': 'Spalte',
        'collapse': 'Zusammenbruch',
        'expand': 'Erweitern'
    },
    "pivotfieldlist": {
        'fieldList': 'Feld Liste',
        'dropRowPrompt': 'Drop Reihe hier',
        'dropColPrompt': 'Drop column Hier',
        'dropValPrompt': 'Drop wert hier',
        'dropFilterPrompt': 'Drop Filter Hier',
        'addPrompt': 'Feld hinzufügen',
        'centerHeader': 'Ziehen Sie die Felder zwischen den
Bereichen unten:',
        'add': 'Hinzufügen',
        'drag': 'Ziehen',
        'filters': 'Filter',
        'rows': 'Zeilen',
        'columns': 'Spalten',
        'values': 'Werte',
        'error': 'Fehler',
        'dropAction': 'Berechnetes Feld nicht in jeder anderen
Region außer Wert Achse sein.',
        'search': 'Suche',
        'close': 'Schließen',
        'cancel': 'Abbrechen',
        'delete': 'Löschen',
        'alert': 'Warnung',
        'warning': 'Warnung',
        'ok': 'OK',
        'allFields': 'Alle Felder',
        'noMatches': 'Keine Treffer'
    }
    }
    });
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
    };
    this.width = "100%";
}
}

```


MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Right-to-left (RTL)

RTL provides an option to switch the text direction and layout of the pivot table 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 pivot table, set the `enableRtl` property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
import { IDataOptions, IDataset, FieldListService, CalculatedFieldService,
GroupingBarService } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService, CalculatedFieldService, GroupingBarService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings [width]=width
allowCalculatedField='true' showGroupingBar='true' locale='ar-AE'
enableRtl= true showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    L10n.load({
      'ar-AE': {
        'pivotview': {
          'grandTotal': 'المجموع الكلي',
          'total': 'المجموع',
          'value': 'القيمة',
          'noValue': 'لا قيمة لها',
          'row': 'صف',
          'column': 'العمود',
          'collapse': 'الانهايار',
          'expand': 'توسيع'
        }
      },
    },
```

```

        'pivotfieldlist': {
            'fieldList': 'قائمة الحقول',
            'dropRowPrompt': 'تراجع الخلاف هنا',
            'dropColPrompt': 'انخفاض العمود هنا',
            'dropValPrompt': 'انخفاض قيمة هنا',
            'dropFilterPrompt': 'انخفاض هنا عامل التصفية',
            'addPrompt': 'اضافة حقل هنا',
            'adaptiveFieldHeader': 'اختر الحقل',
            'centerHeader': 'اسحب المجالات بين المناطق الموضحة',
            'add': 'اضافة',
            'drag': 'اسحب',
            'filters': 'عوامل التصفية',
            'rows': 'الصفوف',
            'columns': 'الاعمدة',
            'values': 'قيم',
            'search': 'البحث',
            'close': 'قريب',
            'cancel': 'الغاء',
            'delete': 'احذف',
            'alert': 'حالة تاهب قصوى',
            'warning': 'تحذير',
            'ok': 'موافق',
            'allFields': 'جميع الحقول',
            'noMatches': 'لا مباريات'
        }
    });
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
    };
    this.width = "100%";
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Internationalization](#)
- [Localization](#)

Accessibility in Angular Pivotview component

The pivot table 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 pivot table 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

[WAI-ARIA](#) (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components. The following ARIA attributes are used in the pivot table component:

| Attributes | Purpose |

| --- | --- |

| **role=grid** | Attribute added to identify the grid component element within the pivot table element. |

| **role=region** | Attribute added to identify the chart component element within the pivot table element. |

| **role=button** | This attribute is added to the pager navigation buttons as well as the buttons in the dialog popup such as field list, calculated field, member editor, conditional formatting of pivot table component to indicate that it is a clickable element. |

| **role=table** | This attribute is added to each conditional formatting style container element to denote it as a table. |

| **role=tableitems** | This attribute is added to the container element that appears inside the number formatting popup to indicate it as a table. |

| **aria-disabled** | The buttons within the dialog popups, such as field list, calculated field and member editor, will be disabled based on their usability. To indicate its disabled state, we will add this attribute with the values **true**. By default, the attribute value is set to **false**. |

| **aria-label** | This attribute is added to label elements that are placed inside the pager, member editor popup, and calculated field popup to identify them as label elements. |

| **aria-selected** | This attribute is added to the selected treeview item in the calculated field popup with the value as **true** to denote that it is a selected element. |

| **aria-colspan** | This attribute is added to the **th** elements in the **e-table**, which represent the column span value. |

| **aria-rowspan** | This attribute is added to the **th** elements in the **e-table**, which represent the row span value. |

| **data-type** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It represents the aggregate type for the specified field. |

| **data-caption** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It represents the caption for the specified field. |

| **data-basefield** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It denotes the base field for the specified field, which is

used to display the values for aggregation types such as **DifferenceFrom**, **PercentageOfDifferenceFrom**, and **PercentageOfParentTotal**. |

| **data-baseitem** | This attribute is added to the treeview item in the calculated field popup, as well as the buttons in the grouping bar and field list. It denotes the base item for the specified field, which is used to display the values for aggregation types such as **DifferenceFrom**, **PercentageOfDifferenceFrom**, and **PercentageOfParentTotal**. |

| **data-field** | This attribute is added to the treeview item in the calculated field popup. It denotes the name of the specified field. |

| **data-membertype** | This attribute is added to the treeview item in the calculated field popup. It denotes the member type of the selected OLAP calculated field. |

| **data-hierarchy** | This attribute is added to the treeview item in the calculated field popup. It denotes the parent hierarchy unique name of the selected OLAP calculated field. |

| **data-formula** | This attribute is added to the treeview item in the calculated field popup. It denotes the formula used for the specified calculated field. |

| **data-formatString** | This attribute is added to the treeview item in the calculated field popup. It denotes the format string used for the specified calculated field. |

| **data-customformatstring** | This attribute is added to the treeview item in the calculated field popup. It denotes the custom format string used for the specified calculated field. |

Keyboard interaction

The pivot table 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 Message component.

| **Press** | **To do this** |

| --- | --- |

| **Tab / Shift + Tab** | To focus the close icon in the message. |

| **Enter / Space** | Closes the focused close icon's message. |

Pivot Table

| **Press** | **To do this** |

| --- | --- |

| **Tab** | Moves the cell focus right side. If no cells are focused, it moves to the next active element in the browser page. |

| **Shift + Tab** | Moves the cell focus left side. If no cells are focused, it moves to the previous active element in the browser page. |

| **DownArrow** | Moves the cell focus downwards. If the selection is enabled in the pivot table, then it will move downwards to select next row or column or individual cell. |

| **UpArrow** | Moves the cell focus upwards. If the selection is enabled in the pivot table, then it will move upwards to select previous row or column or individual cell. |

| LeftArrow | Moves the cell focus left side. If the selection is enabled in the pivot table, then it will move left side to select previous row or column or individual cell. |

| RightArrow | Moves the cell focus right side. If the selection is enabled in the pivot table, then it will move right side to select next row or column or individual cell. |

| Shift + DownArrow | Extends the cell selection downwards. |

| Shift + UpArrow | Extends the cell selection selection upwards. |

| Shift + LeftArrow | Extends the cell selection to the left side. |

| Shift + RightArrow | Extends the cell selection to the right side. |

| Ctrl + A | Selects all cells. |

| Esc | Deselects all cells. If the current active element is a context menu, then the context menu popup will be closed. |

| Home | Goes to the first cell in the current row. |

| End | Goes to the last cell in the current row. |

| Ctrl + Home | Goes to the first cell in the table. |

| Ctrl + End | Goes to the last cell in the table. |

| Enter | If the current cell is an expand/collapse cell, it performs expand/collapse operation (drill operation). If the current row/column header is in value sort state, it performs value sorting. If the current cell is in selection state, it moves to the next row, column or individual cell. If drill-through or editing is enabled in the pivot table, the drill-through dialog will be opened based on the selected value cell. If the current active element is a context menu popup, menu selection will be performed. |

| Shift + Enter | If value sorting is enabled in the pivot table and the current cell is a header with respect to its value axis, it performs value sorting to either ascending or descending order. If the current cell is in selection state, it moves to the previous row, column or individual cell. |

| Ctrl + Enter | If hyperlink is enabled in the current cell, it performs hyperlink selection. |

| Shift + F10 or Menu | If context menu is enabled in the pivot table, the context menu popup will be opened in the current cell. |

Field List

| Press | To do this |

| --- | --- |

| Shift + Ctrl + F | If the popup field list is enabled in either the pivot table or the pivot chart, the field list dialog will be opened. |

| Tab | Moves to the next active element in the field list. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the field list. If no active elements present, it moves to the previous active element in the browser page. |

| Shift + F | If the current active element is a field's button and if it has a filter icon, the filter dialog will open to perform filtering. |

| Shift + S | If the current active element is a field's button and if it has a sort icon, the sorting will be performed to the selected field. |

| Shift + E | If the current active element is a calculated field's button and if it has an edit icon, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Enter | Performs the selection operation of the current active element. If the current active element is a field's button and it has a dropdown icon, the aggregation menu will open to perform calculations using aggregation options to the selected value field. |

| Delete | If the current active element is a field's button, the selected field will be removed from the current report. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. |

| RightArrow | If the current active element is a tree node, it expands the current node. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| Esc or Escape | Closes the popup field list dialog. |

Grouping Bar

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element (field's button) in the grouping bar. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element (field's button) in the grouping bar. If no active elements present, it moves to the previous active element in the browser page. |

| Shift + F | If the current active element is a field's button and if it has a filter icon, the filter dialog will be opened to perform filtering. |

| Shift + S | If the current active element is a field's button and if it has a sort icon, the sorting will be performed to the selected field. |

| Shift + E | If the current active element is a calculated field's button and if it has an edit icon, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Enter | Performs the selection operation of the current active element. If the current active element is a field's button and if it has a dropdown icon, the aggregation menu will be opened to perform calculations using aggregation options to the selected value field. |

| Delete | If the current active element is a field's button, the selected field will be removed from the current report. |

| DownArrow | If the current active element is a dropdown list, the next item will be selected. |

| UpArrow | If the current active element is a dropdown list, the previous item will be selected. |

| Home | If the current active element is a dropdown list, the first item will be selected. |

| End | If the current active element is a dropdown list, the last item will be selected. |

| Alt + Down | If the current active element is a dropdown list, the popup will be opened. |

| Alt + Down | If the current active element is a dropdown list, the popup will be closed. |

| Esc or Escape | Closes the dropdown list.

Filter Dialog

| Press | To do this |

| --- | --- |

| Shift + F | If the current active element is a field's button and if it has a filter icon in either the field list or grouping bar UI, the filter dialog will be opened to perform filtering. |

| Tab | Moves to the next active element in the filter dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the filter dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. If the current active element is a tab, it moves focus to the previous tab element. |

| RightArrow | If the current active element is a tree node, it expands the current node. If the current active element is a tab, it moves focus to the next tab element. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| Alt + Down | If the current active element is a DropDownList or DatePicker or DateTimePicker, the popup will be opened. |

| Alt + Up | If the current active element is a DropDownList or DatePicker or DateTimePicker, the popup will be closed. |

| Enter | Performs the selection operation of the current active element. If the current active element is a tab, the current tab element will be selected. If the current active element is a tree

node, the current node will be either checked or unchecked. If the current active element is DropDownList, the focus item will be selected, and the popup list will close when it is open. Otherwise, toggles the popup list. |

| Esc or Escape | Closes the filter dialog. |

Calculated Field Dialog

| **Press** | **To do this** |

| --- | --- |

| Shift + E | If the current active element is a field's button and if it has an edit icon in either the field list or grouping bar UI, the calculated field dialog will be opened to perform editing the selected calculated field. |

| Tab | Moves to the next active element in the calculated field dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the calculated field dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a tree node, it moves to the next node. |

| UpArrow | If the current active element is a tree node, it moves to the previous node. |

| LeftArrow | If the current active element is a tree node, it collapses the current node. |

| RightArrow | If the current active element is a tree node, it expands the current node. If the current active element is a tree node and has a menu icon, the aggregation menu will be opened to select appropriate aggregation type to the selected field. |

| Home | If the current active element is a tree node, it goes to the first node. |

| End | If the current active element is a tree node, it goes to the last node. |

| Enter | Performs the selection operation of the current active element. If the current active element is a tree node, it copies the selected field name/formula to the formula text area to perform calculations. |

| Esc or Escape | Closes the calculated field dialog. |

Formatting Dialog

| **Press** | **To do this** |

| --- | --- |

| Tab | Moves to the next active element in the formatting dialog. If no active elements present, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the formatting dialog. If no active elements present, it moves to the previous active element in the browser page. |

| DownArrow | If the current active element is a DropDownList, the next item will be selected. |

|

- | UpArrow | If the current active element is a DropDownList, the previous item will be selected. |
- | Home | If the current active element is a DropDownList, the first item will be selected. |
- | End | If the current active element is a DropDownList, the last item will be selected. |
- | Alt + Down | If the current active element is a DropDownList or ColorPicker, the popup will be opened. |
- | Alt + Down | If the current active element is a DropDownList or ColorPicker, the popup will be closed. |
- | Enter | Performs the selection operation of the current active element. |
- | Esc or Escape | Closes the formatting dialog. |

Toolbar

- | Press | To do this |
- | --- | --- |
- | Tab | Moves to the next active option in the toolbar. If no active elements present, it moves to the next active element in the browser page. |
- | Shift + Tab | Moves to the previous active option in the toolbar. If no active elements present, it moves to the previous active element in the browser page. |
- | Enter | Performs the selection operation of the current active element. |

Drill-Through Dialog

- | Press | To do this |
- | --- | --- |
- | Tab | Moves to the next active element in the drill-through dialog. If the current active element is a Grid cell, it moves the cell focus to right side. If no active elements present, then it moves to the next active element in the browser page. |
- | Shift + Tab | Moves to the previous active element in the drill-through dialog. If the current active element is a Grid cell, it moves the cell focus to left side, If no active elements present, then it moves to the previous active element in the browser page. |
- | DownArrow | Moves the row/cell focus downwards. |
- | UpArrow | Moves the row/cell focus upwards. |
- | LeftArrow | Moves the cell focus left side. |
- | RightArrow | Moves the cell focus right side. |
- | Home | Goes to the first cell in the current row. |
- | End | Goes to the last cell in the current row. |
- | Ctrl + Home | Goes to the first cell in the table. |
- | Ctrl + End | Goes to the last cell in the table. |

| Enter | Performs the selection operation of the current active element. |

| Esc or Escape | If the cell is in selected state, the it deselects all rows/cells. If the row/cell is in edit state, it cancels the current entries in the row/cell. If the current active element is not a row/cell, it closes the drill-through dialog. |

| F2 | Initiate editing a row/cell in the data grid. |

| Insert | Adds a new row/cell in the data grid. |

| Delete | Removes the selected row in the data grid. |

Some commonly used applicable key combinations and their relative functionalities in all dialogs are listed below.

| Press | To do this |

| --- | --- |

| Tab | Moves to the next active element in the dialog. If either no active elements present in the dialog or an overlay is not present in the dialog, it moves to the next active element in the browser page. |

| Shift + Tab | Moves to the previous active element in the dialog. If either no active elements present in the dialog or an overlay is not present in the dialog, it moves to the previous active element in the browser page. |

| Space | If the current active element is a tree node or a checkbox element, it will be either checked or unchecked. |

| 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 be triggered. The Enter key will not be worked, when the dialog content contains any text area with initial focus. |

| Esc or Escape | Closes the dialog. |

Ensuring accessibility

The pivot table component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the pivot table component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the pivot table component with accessibility tools.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  NumberFormattingService,
```

```

    ToolbarService, ConditionalFormattingService, ToolbarItems,
    DisplayOption, IDataset, DataSourceSettings,
    GroupingBarSettings, CellEditSettings, GroupingBarService,
    GroupingService
} from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [
    CalculatedFieldService, ToolbarService, ConditionalFormattingService,
    FieldListService, NumberFormattingService, GroupingBarService,
    GroupingService
  ],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings
  allowExcelExport='true' allowNumberFormatting='true'
allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true'
  allowCalculatedField='true' showFieldList='true' width='100%'
[displayOption]='displayOption' height='450' [toolbar]='toolbarOptions'
  (saveReport)='saveReport($event)' (loadReport)='loadReport($event)'
(fetchReport)='fetchReport($event)' (renameReport)='renameReport($event)'
  (removeReport)='removeReport($event)' (newReport)='newReport()'
showGroupingBar='true' [groupingBarSettings]='groupingBarSettings'
  [chartSettings]='chartSettings' allowGrouping='true'
allowDeferLayoutUpdate='true' (toolbarRender)='toolbarRender($event)'>
</ejs-pivotview></div>`
})
export class AppComponent {
  public dataSourceSettings?: IDataset;
  public toolbarOptions?: ToolbarItems[];
  public displayOption?: DisplayOption;
  public chartSettings?: ChartSettings;
  public groupingBarSettings?: GroupingBarSettings;
  public editSettings?: CellEditSettings;
  public gridSettings?: GridSettings;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  saveReport(args: any) {
    let reports = [];
    let isSaved: boolean = false;
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== '') {
      reports = JSON.parse(localStorage['pivotviewReports']);
    }
    if (args.report && args.reportName && args.reportName !== '') {
      reports.map(function (item: any): any {

```

```

        if (args.reportName === item.reportName) {
            item.report = args.report; isSaved = true;
        }
    });
    if (!isSaved) {
        reports.push(args);
    }
    localStorage['pivotviewReports'] = JSON.stringify(reports);
}

fetchReport(args: any) {
    let reportCollection: string[] = [];
    let reeportList: string[] = [];
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
    }
    reportCollection.map(function (item: any): void {
reeporList.push(item.reportName); });
    args.reportName = reeportList;
}

loadReport(args: any) {
    let reportCollection: string[] = [];
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
    }
    reportCollection.map(function (item: any): void {
        if (args.reportName === item.reportName) {
            args.report = item.report;
        }
    });
    if (args.report) {
        (this.pivotGridObj!.dataSourceSettings as DataSourceSettings) =
JSON.parse(args.report).dataSourceSettings;
    }
}

removeReport(args: any) {
    let reportCollection: any[] = [];
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        reportCollection = JSON.parse(localStorage['pivotviewReports']);
    }
    for (let i: number = 0; i < reportCollection.length; i++) {
        if (reportCollection[i].reportName === args.reportName) {
            reportCollection.splice(i, 1);
        }
    }
    if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
        localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
    }
}

renameReport(args: any) {
    let reportCollection: string[] = [];

```

```

        if (localStorage['pivotviewReports'] &&
        localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        reportCollection.map(function (item: any): any { if (args.reportName
        === item.reportName) { item.reportName = args.rename; } });
        if (localStorage['pivotviewReports'] &&
        localStorage['pivotviewReports'] !== "") {
            localStorage['pivotviewReports'] =
            JSON.stringify(reportCollection);
        }
    }
    toolbarRender(args: any) {
        args.customToolbar.splice(6, 0, {
            type: 'Separator'
        });
        args.customToolbar.splice(9, 0, {
            type: 'Separator'
        });
    }
    newReport() {
        this.pivotGridObj?.setProperties({ dataSourceSettings: { columns:
        [], rows: [], values: [], filters: [] } }, false);
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
        'Load',
            'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
        'ConditionalFormatting', 'NumberFormatting', 'FieldList'] as ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: true,
            enableSorting: true,
            allowLabelFilter: true,
            allowValueFilter: true,
            sortSettings: [{ name: 'company', order: 'Descending' }],
            formatSettings: [{ name: 'balance', format: 'C' }, { name:
        'date', format: 'dd/MM/yyyy-hh:mm', type: 'date' }],
            drilledMembers: [{ name: 'product', items: ['Bike', 'Car'] }, {
        name: 'gender', items: ['male'] }],
            filterSettings: [
                { name: 'date', type: 'Date', condition: 'Between', value1:
        new Date('02/16/2000'), value2: new Date('02/16/2002') },
                { name: 'age', type: 'Number', condition: 'Between', value1:
        '25', value2: '35' },
                { name: 'eyeColor', type: 'Exclude', items: ['blue'] }
            ],
            rows: [{ name: 'state' }, { name: 'eyeColor' }],
            columns: [{ name: 'gender', caption: 'Population' }, { name:
        'isActive' }],
            values: [{ name: 'balance' }, { name: 'quantity' }],
            filters: [],
            conditionalFormatSettings: [
                {
                    measure: 'balance',
                    value1: 100000,

```

```

        conditions: 'LessThan',
        style: {
            backgroundColor: '#80cbc4',
            color: 'black',
            fontFamily: 'Tahoma',
            fontSize: '12px'
        }
    },
    {
        value1: 10,
        value2: 20,
        measure: 'quantity',
        conditions: 'Between',
        style: {
            backgroundColor: '#f48fb1',
            color: 'black',
            fontFamily: 'Tahoma',
            fontSize: '12px'
        }
    }
]
};
this.groupingBarSettings = {
    showFieldsPanel: true
} as GroupingBarSettings;
this.chartSettings = {
    value: 'Amount', enableExport: true, chartSeries: { type:
'Column', animation: { enable: false } }, enableMultipleAxis: false
} as ChartSettings;
this.editSettings = {
    allowEditing: true, allowAdding: true, allowDeleting: true,
mode: 'Normal'
} as CellEditSettings;
this.gridSettings = {
    columnWidth: 140,
    contextMenuItems: [
        'Aggregate', 'CalculatedField', 'Drillthrough', 'Excel
Export', 'Pdf Export',
        'Csv Export', 'Expand', 'Collapse', 'Sort Ascending', 'Sort
Descending'
    ]
} as GridSettings
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See also

- [Accessibility in Syncfusion Angular components](#)

Ej1 api migration in Angular Pivotview component

This article describes the API migration process of pivot table component from Essential JS 1 to Essential JS 2.

Data Binding

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| DataSource | **property:** dataSource

<ej-pivotgrid [dataSource.data]="data" [dataSource.rows]="rows" [dataSource.columns]="columns" [dataSource.values]="values"></ej-pivotgrid>

export class PivotGridComponent {
public data; rows; columns; values; filters;
constructor() {
this.data = [{ Amount: 100, Country: "Canada", Date: "FY 2005", Product: "Bike", Quantity: 2, State: "Alberta" }];
this.rows = [{ fieldName: "Country", fieldCaption: "Country" }];
this.columns = [{ fieldName: "Product", fieldCaption: "Product" }];
this.values = [{ fieldName: "Amount", fieldCaption: "Amount" }];
}} | **property:** dataSourceSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
dataSource: this.pivotData,
columns: [{ name: 'Year', caption: 'Production Year' }, { name: 'Quarter' }],
values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount', caption: 'Sold Amount' }],
rows: [{ name: 'Country' }, { name: 'Products' }]];
}} |

| Rows | **property:** rows

<ej-pivotgrid [dataSource.rows]="rows"></ej-pivotgrid>

export class PivotGridComponent {
public rows;
constructor() {
this.rows = [{ fieldName: "Country", fieldCaption: "Country" }];
}} | **property:** rows

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
rows: [{ name: 'Country' }, { name: 'Products' }]];
}} |

| Columns | **property:** columns

<ej-pivotgrid [dataSource.columns]="columns"></ej-pivotgrid>

export class PivotGridComponent {
public columns;
constructor() {
this.columns = [{ fieldName: "Product", fieldCaption: "Product" }];
}} | **property:** columns

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
columns: [{ name: 'Year' }, { name: 'Production Year' }]];
}} |

| Values | **property:** values

<ej-pivotgrid [dataSource.values]="values"></ej-pivotgrid>

export class PivotGridComponent {
public values;
constructor() {
this.values = [{ fieldName: "Amount", fieldCaption: "Amount" }];
}} | **property:** values

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings:


```
IDataOptions;<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>values: [{ name:
'Amount' }]};<br/>}}

|Filters |property: filters<br/><br/><ej-pivotgrid [dataSource.filters]="filters"></ej-
pivotgrid><br/><br/>export class PivotGridComponent {<br/>public filters;<br/>constructor()
{<br/>this.filters= [{ fieldName: "Product", fieldCaption: "Product" }]};<br/>}}|property:
filters<br/><br/><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview><br/><br/>export class
AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>filters: [{ name: 'Year'
},{ name: 'Production Year' }]};<br/>}}|

|Value axis position|Not Applicable|property: valueAxis<br/><br/><ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview><br/><br/>export
class AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>valueAxis:
'row';<br/>}}|
```

Aggregation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

```
|Summary Type|property: summaryType<br/><br/><ej-pivotgrid
[dataSource.values]="values"></ej-pivotgrid><br/><br/>export class PivotGridComponent
{<br/>public values;<br/>constructor() {<br/>this.values= [{ fieldName: "Amount", fieldCaption:
"Amount", summaryType: ej.PivotAnalysis.SummaryType.Count }]};<br/>}}|property:
type<br/><br/><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview><br/><br/>export class
AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions;<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>values: [{ name:
'Amount', type: 'Count' }]};<br/>}}|
```

Number Format

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

```
|Format settings|property: format<br/><br/><ej-pivotgrid [dataSource.values]="values"></ej-
pivotgrid><br/><br/>export class PivotGridComponent {<br/>public values;<br/>constructor()
{<br/>this.values= [{ fieldName: "Amount", fieldCaption: "Amount", format: "currency"
}]};<br/>}}|property: formatSettings<br/><br/><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview><br/><br/>export class
AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions;<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>formatSettings: [{
name: 'balance', format: 'C' },<br/>{ name: 'date', format: 'dd/MM/yyyy-hh:mm', type: 'date'
}]};<br/>}}|
```

Summary Customization

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Show/hide grand totals | **property:** enableGrandTotal

<ej-pivotgrid
[dataSource.enableGrandTotal]="false"></ej-pivotgrid> | **property:**
showGrandTotals

<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public dataSourceSettings:
IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
showGrandTotals:
false};
}} |

| Show/hide row grand totals | **property:** enableRowGrandTotal

<ej-pivotgrid
[dataSource.enableRowGrandTotal]="false"></ej-pivotgrid> | **property:**
showRowGrandTotals

<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public dataSourceSettings:
IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
showRowGrandTotals:
false};
}} |

| Show/hide column grand totals | **property:** enableColumnGrandTotal

<ej-pivotgrid
[dataSource.enableColumnGrandTotal]="false"></ej-pivotgrid> | **property:**
showColumnGrandTotals

<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public dataSourceSettings:
IDataOptions;
ngOnInit(): void {
this.dataSourceSettings =
{
showColumnGrandTotals: false};
}} |

| Show/hide sub-totals | Not Applicable | **property:** showSubTotals

<ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export
class AppComponent implements OnInit {
public dataSourceSettings:
IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
showSubTotals:
false};
}} |

| Show/hide row sub-totals | Not Applicable | **property:** showRowSubTotals

<ejs-pivotview
#pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-
pivotview>

export class AppComponent implements OnInit {
public
dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings =
{
showRowSubTotals: false};
}} |

| Show/hide column sub-totals | Not Applicable | **property:** showColumnSubTotals

<ejs-
pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-
pivotview>

export class AppComponent implements OnInit {
public
dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings =
{
showColumnSubTotals: false};
}} |

| Show/hide sub-totals for specific field | **property:** showSubTotal

<ej-pivotgrid
[dataSource.rows]="rows"></ej-pivotgrid>

export class PivotGridComponent

```
{<br/>public rows;<br/>constructor() {<br/>this.rows= [{ name: 'company', showSubTotal: false
}];<br/>}} <br/>}; | property: showSubTotals<br/><br/><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview><br/><br/>export class
AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions;<br/>ngOnInit(): void {<br/>this.dataSourceSettings = {<br/>rows: [{ name:
'company', showSubTotals: false }]<br/>};<br/>}}|
```

Drill operation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Expand All | **property:** enableCollapseByDefault

<ej-pivotgrid
[enableCollapseByDefault]="true"></ej-pivotgrid> | **property:** expandAll

<ejs-pivotview
#pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-
pivotview>

export class AppComponent implements OnInit {
public
dataSourceSettings: IDataOptions
ngOnInit(): void {
this.dataSourceSettings =
{
expandAll: false;
}}|

| Drill Up/Down | **property:** collapsedMembers

<ej-pivotgrid
[collapsedMembers]="collapsedMembers"></ej-pivotgrid>

export class
PivotGridComponent {
public collapsedMembers;
constructor()
{
collapsedMembers: { Country: ["Canada", "France"] };
}} | **property:**
drilledMembers

<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public dataSourceSettings:
IDataOptions
ngOnInit(): void {
this.dataSourceSettings = {
drilledMembers: [{
name: 'Country', items: ['France'] }];
}}|

Field List

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Show/hide field list | Not Applicable | **property:** showFieldList

<ejs-pivotview #pivotview
id='PivotView' showFieldList='true'></ejs-pivotview>|

| Defer update | **property:** enableDeferUpdate

<ej-pivotgrid
[enableDeferUpdate]="true"></ej-pivotgrid> | Not Applicable |

| Control initialization | **component:** PivotSchemaDesigner

<ej-pivotschemadesigner
id="PivotSchemaDesigner1"></ej-pivotschemadesigner> | **component:**
PivotFieldList

<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList'
[dataSourceSettings]=dataSourceSettings></ejs-pivotfieldlist>|

| Render mode | Not Applicable | **property:** renderMode

<ejs-pivotfieldlist #pivotfieldlist
id='PivotFieldList' [dataSourceSettings]=dataSourceSettings renderMode="Fixed"></ejs-
pivotfieldlist>|

| Show/hide calculated field button | Not Applicable | **property:** allowCalculatedField

<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList' [dataSourceSettings]=dataSourceSettings allowCalculatedField="true"></ejs-pivotfieldlist> |

| Show/hide values button | Not Applicable | **property:** showValuesButton

<ejs-pivotfieldlist #pivotfieldlist id='PivotFieldList' [dataSourceSettings]=dataSourceSettings showValuesButton="true"></ejs-pivotfieldlist> |

Grouping Bar

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Show/hide Grouping bar | **property:** enableGroupingBar

<ej-pivotgrid [enableGroupingBar]="true"></ej-pivotgrid> | **property:** showGroupingBar

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings showGroupingBar='true'></ejs-pivotview> |

| Grouping Bar Settings | Not Applicable | **property:** groupingBarSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings showGroupingBar='true'[groupingBarSettings]='groupingSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public groupingSettings: GroupingBarSettings;
ngOnInit(): void {
this.groupingSettings = {
showFilterIcon: false
} as GroupingBarSettings;
}} |

| Show/hide values button | Not Applicable | **property:** showValuesButton

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings showValuesButton='true'></ejs-pivotview> |

Filtering

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Filter settings | **property:** filterItems

<ej-pivotgrid [dataSource.columns]="columns"></ej-pivotgrid>

export class PivotGridComponent {
public columns;
constructor() {
this.columns= [{ fieldName: "Product", fieldCaption: "Product", filterItems: {
filterType: ej.PivotAnalysis.FilterType.Include,
values: ["Bike", "Car"]} }];
}} | **property:** filterSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
filterSettings: [{ name: 'Country', type: 'Exclude', items: ['United States'] }];
}} |

Maximum node limit in member editor

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Max node limit in member editor | **property:** maxNodeLimitInMemberEditor

<ej-pivotgrid [maxNodeLimitInMemberEditor]="100"></ej-pivotgrid> | **property:** maxNodeLimitInMemberEditor

<ejs-pivotview #pivotview id='PivotView' maxNodeLimitInMemberEditor='100'></ejs-pivotview> |

No Data Items

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Show/hide "no data" items | Not Applicable | **property:** showNoDataItems

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
rows: [{ name: 'company', showNoDataItems: true }];
}} |

Advanced filtering

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Label filtering | **property:** enableAdvancedFilter

<ej-pivotgrid [dataSource.rows]="rows" [enableAdvancedFilter]="true"></ej-pivotgrid>

export class PivotGridComponent {
public rows;
constructor() {
this.rows= [{ fieldName: "Country", fieldCaption: "Country", advancedFilter : [{
labelFilterOperator: ej.olap.LabelFilterOptions.EndsWith,
values: ["es"]}]}
}} | **property:** allowLabelFilter

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
allowLabelFilter: true,
filterSettings: [{
name: 'product',
type: 'Label',
condition: 'Between',
value1: 'e',
value2: 'v' }]
}} |

| Value filtering | **property:** enableAdvancedFilter

<ej-pivotgrid [dataSource.rows]="rows" [enableAdvancedFilter]="true"></ej-pivotgrid>

export class PivotGridComponent {
public rows;
constructor() {
this.rows= [{ fieldName: "Country", fieldCaption: "Country", advancedFilter:[{
valueFilterOperator: ej.olap.ValueFilterOptions.GreaterThan,
values: ["200"]}]}
}} | **property:** allowValueFilter

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
allowValueFilter: true,
filterSettings: [{
name: 'product',
measure: 'quantity',
type: 'Value',
condition: 'Between',
value1: '3250' }]
}} |

Drill Through

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Show/hide drill though feature | **property:** enableDrillThrough

<ej-pivotgrid
[enableDrillThrough]="true"></ej-pivotgrid> | **property:** allowDrillThrough

<ejs-pivotview
#pivotview id='PivotView' allowDrillThrough='true'></ejs-pivotview> |

| Event Triggers when cell clicked in pivot table widget | **event:** drillThrough

<ej-pivotgrid
(drillThrough)="onDrillThrough(\$event)"></ej-pivotgrid>

export class
PivotGridComponent {
onDrillThrough(args) {}
} | **event:** drillThrough

<ejs-
pivotview #pivotview id='PivotView' (drillThrough)='onDrillThrough(\$event)'></ejs-
pivotview>

export class AppComponent {
onDrillThrough(args): void {}
}

Cell Editing

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Edit settings | Not Applicable | **property:** editSettings

<ejs-pivotview #pivotview
id='PivotView' [editSettings]=editSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public editSettings: CellEditSettings;
ngOnInit():
void {
this.editSettings = {};
}}

| Show/hide cell editing feature | **property:** enableCellEditing

<ej-pivotgrid
[enableCellEditing]="true"></ej-pivotgrid> | **property:** allowEditing

<ejs-pivotview
#pivotview id='PivotView' [editSettings]=editSettings></ejs-pivotview>

export class
AppComponent implements OnInit {
public editSettings: CellEditSettings;
ngOnInit():
void {
this.editSettings = {
allowAdding: true, allowDeleting: true, allowEditing: true,
mode: 'Normal'
};
}}

Hyperlink

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Hyperlink settings | **property:** hyperlinkSettings

<ej-pivotgrid
[hyperlinkSettings]="hyperlinkSettings"></ej-pivotgrid>

export class
PivotGridComponent {
public hyperlinkSettings;
constructor()
{
this.hyperlinkSettings = {};
}} | **property:** hyperlinkSettings

<ejs-pivotview
#pivotview id='PivotView' [hyperlinkSettings]=hyperlinkSettings></ejs-
pivotview>

export class AppComponent implements OnInit {
public
hyperlinkSettings: HyperLinkSettings;
ngOnInit(): void {
this.hyperlinkSettings =
{
};
}}

| Show/hide hyperlink to all cells | Not Applicable | **property:** showHyperlink

<ejs-pivotview
#pivotview id='PivotView' [hyperlinkSettings]=hyperlinkSettings></ejs-
pivotview>

export class AppComponent implements OnInit {
public
hyperlinkSettings: HyperLinkSettings;
ngOnInit(): void {
this.hyperlinkSettings =
{
showHyperlink: 'true'
};
}}

| Show/hide hyperlink to row headers | **property:** enableRowHeaderHyperlink

<ej-pivotgrid
[hyperlinkSettings]="hyperlinkSettings"></ej-pivotgrid>

export class
PivotGridComponent {
public hyperlinkSettings;
constructor()


```
{<br/>this.hyperlinkSettings= { enableRowHeaderHyperlink: "true"};<br/>}} | property:
showRowHeaderHyperlink<br/><br/><ejs-pivotview #pivotview id='PivotView'
[hyperlinkSettings]=hyperlinkSettings></ejs-pivotview><br/><br/>export class AppComponent
implements OnInit {<br/>public hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void
{<br/>this.hyperlinkSettings = {<br/>showRowHeaderHyperlink: 'true'<br/>};<br/>}}|
```

```
| Show/hide hyperlink to column headers | property: enableColumnHeaderHyperlink<br/><br/><ej-
pivotgrid [hyperlinkSettings]="hyperlinkSettings"></ej-pivotgrid><br/><br/>export class
PivotGridComponent {<br/>public hyperlinkSettings;<br/>constructor()
{<br/>this.hyperlinkSettings= { enableColumnHeaderHyperlink: "true"};<br/>}} | property:
showColumnHeaderHyperlink<br/><br/><ejs-pivotview #pivotview id='PivotView'
[hyperlinkSettings]=hyperlinkSettings></ejs-pivotview><br/><br/>export class AppComponent
implements OnInit {<br/>public hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void
{<br/>this.hyperlinkSettings = {<br/>showColumnHeaderHyperlink: 'true'<br/>};<br/>}}|
```

```
| Show/hide hyperlink to value cells | property: enableValueCellHyperlink<br/><br/><ej-pivotgrid
[hyperlinkSettings]="hyperlinkSettings"></ej-pivotgrid><br/><br/>export class
PivotGridComponent {<br/>public hyperlinkSettings;<br/>constructor()
{<br/>this.hyperlinkSettings= { enableValueCellHyperlink: "true"};<br/>}} | property:
showValueCellHyperlink<br/><br/><ejs-pivotview #pivotview id='PivotView'
[hyperlinkSettings]=hyperlinkSettings></ejs-pivotview><br/><br/>export class AppComponent
implements OnInit {<br/>public hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void
{<br/>this.hyperlinkSettings = {<br/>showValueCellHyperlink: 'true'<br/>};<br/>}}|
```

```
| Show/hide hyperlink to summary cells | property: enableSummaryCellHyperlink<br/><br/><ej-
pivotgrid [hyperlinkSettings]="hyperlinkSettings"></ej-pivotgrid><br/><br/>export class
PivotGridComponent {<br/>public hyperlinkSettings;<br/>constructor()
{<br/>this.hyperlinkSettings= { enableSummaryCellHyperlink: "true"};<br/>}} | property:
showSummaryCellHyperlink<br/><br/><ejs-pivotview #pivotview id='PivotView'
[hyperlinkSettings]=hyperlinkSettings></ejs-pivotview><br/><br/>export class AppComponent
implements OnInit {<br/>public hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void
{<br/>this.hyperlinkSettings = {<br/>showSummaryCellHyperlink: 'true'<br/>};<br/>}}|
```

```
| Show/hide hyperlink using specific conditions | Not Applicable | property:
conditionalSettings<br/><br/><ejs-pivotview #pivotview id='PivotView'
[hyperlinkSettings]=hyperlinkSettings></ejs-pivotview><br/><br/>export class AppComponent
implements OnInit {<br/>public hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void
{<br/>this.hyperlinkSettings = {<br/>conditionalSettings: [{<br/>measure: 'Units Sold',
conditions: 'Between', value1: 150, value2: 200<br/>}];<br/>}}|
```

```
| Show/hide hyperlink for row or column | Not Applicable | property: headerText<br/><br/><ejs-
pivotview #pivotview id='PivotView' [hyperlinkSettings]=hyperlinkSettings></ejs-
pivotview><br/><br/>export class AppComponent implements OnInit {<br/>public
hyperlinkSettings: HyperLinkSettings;<br/>ngOnInit(): void {<br/>this.hyperlinkSettings =
{<br/>headerText: 'FY 2015.Q1.Units Sold'<br/>};<br/>}}|
```

| Event Triggers when row headers clicked in pivot table widget | **event:**
 rowHeaderHyperlinkClick

<EJ.PivotGrid id="PivotGrid" rowHeaderHyperlinkClick=
 "onRowHeaderHyperlinkClick"></EJ.PivotGrid>

function
 onRowHeaderHyperlinkClick({}) | **event:** hyperlinkCellClick

<ejs-pivotview #pivotview
 id='PivotView' (hyperlinkCellClick)='onHyperlinkCellClick(\$event)'></ejs-
 pivotview>

export class AppComponent {
onHyperlinkCellClick(args): void
 {}
}|

| Event Triggers when column headers clicked in pivot table widget | **event:**
 columnHeaderHyperlinkClick

<ej-pivotgrid [dataSource]="dataSource"
 (columnHeaderHyperlinkClick)="onColumnHeaderHyperlinkClick(\$event)"></ej-
 pivotgrid>

export class PivotGridComponent
 {
onColumnHeaderHyperlinkClick(args) {}
} | **event:** hyperlinkCellClick

<ejs-
 pivotview #pivotview id='PivotView' (hyperlinkCellClick)='onHyperlinkCellClick(\$event)'></ejs-
 pivotview>

export class AppComponent {
onHyperlinkCellClick(args): void
 {}
}|

| Event Triggers when value cells clicked in pivot table widget | **event:**
 valueCellHyperlinkClick

<ej-pivotgrid [dataSource]="dataSource"
 (valueCellHyperlinkClick)="onValueCellHyperlinkClick(\$event)"></ej-pivotgrid>

export
 class PivotGridComponent {
onValueCellHyperlinkClick(args) {}
} | **event:**
 hyperlinkCellClick

<ejs-pivotview #pivotview id='PivotView'
 (hyperlinkCellClick)='onHyperlinkCellClick(\$event)'></ejs-pivotview>

export class
 AppComponent {
onHyperlinkCellClick(args): void {}
}|

| Event Triggers when summary cells clicked in pivot table widget | **event:**
 summaryCellHyperlinkClick

<ej-pivotgrid [dataSource]="dataSource"
 (summaryCellHyperlinkClick)="onSummaryCellHyperlinkClick(\$event)"></ej-
 pivotgrid>

export class PivotGridComponent {
onSummaryCellHyperlinkClick(args)
 {}
} | **event:** hyperlinkCellClick

<ejs-pivotview #pivotview id='PivotView'
 (hyperlinkCellClick)='onHyperlinkCellClick(\$event)'></ejs-pivotview>

export class
 AppComponent {
onHyperlinkCellClick(args): void {}
}|

Defer Layout Update

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Show/hide defer layout update | **property:** enableDeferUpdate

<ej-pivotgrid
 [enableDeferUpdate]="true"></ej-pivotgrid> | **property:** allowDeferLayoutUpdate

<ejs-
 pivotview #pivotview id='PivotView' allowDeferLayoutUpdate='true'></ejs-pivotview> |

Sorting

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Enable/disable sorting | Not Applicable | **property:** enableSorting

<ejs-pivotview #pivotview
 id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export
 class AppComponent implements OnInit {
public dataSourceSettings:

IDataOptions
ngOnInit(): void {
this.dataSourceSettings = {
enableSorting: false;
}}

| Sort settings | **property:** sortOrder

<ej-pivotgrid [dataSource.rows]="rows"></ej-pivotgrid>

export class PivotGridComponent {
public rows;
constructor() {
this.rows= [{
fieldName: "Country",
fieldCaption: "Country",
sortOrder: ej.PivotAnalysis.SortOrder.Descending
}};
}}
| **property:** sortSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
sortSettings: [{
name: 'company',
order: 'Descending' }]};
}}|

Value Sorting

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Enable/disable value sorting | Not Applicable | **property:** enableValueSorting

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings enableValueSorting='true'></ejs-pivotview>|

| Value sort settings | **property:** valueSortSettings

<ej-pivotgrid [valueSortSettings]="valueSortSettings"></ej-pivotgrid>

export class PivotGridComponent {
public valueSortSettings;
constructor() {
this.valueSortSettings= {
headerText: "Bike##Quantity",
headerDelimiters: "##",
sortOrder: ej.PivotAnalysis.SortOrder.Descending
}};
}}| **property:** valueSortSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings></ejs-pivotview>

export class AppComponent implements OnInit {
public dataSourceSettings: IDataOptions;
ngOnInit(): void {
this.dataSourceSettings = {
valueSortSettings: {
headerText: 'FY 2015##Sold Amount',
headerDelimiter: '##',
sortOrder: 'Descending' } };
}}|

Calculated Field

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Show/hide calculated field | Not Applicable | **property:** allowCalculatedField

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings allowCalculatedField='true'></ejs-pivotview>|

| Calculated field settings | **property:** values

<ej-pivotgrid [dataSource.values]="values"></ej-pivotgrid>

export class PivotGridComponent {
public values;
constructor() {
this.values=[{
fieldName:"Amount",
fieldCaption:"Amount"
},
{
fieldName:"Price",
fieldCaption: "Price",
isCalculatedField: true,
formula: "Amount*15"};
}}| **property:** calculatedFieldSettings

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings showFieldList='true'

```
allowCalculatedField='true'></ejs-pivotview><br/><br/>export class AppComponent  
implements OnInit {<br/>public dataSourceSettings: IDataOptions;<br/>ngOnInit(): void  
{<br/>this.dataSourceSettings = {<br/>values: [{<br/>name: 'price', type: 'CalculatedField'  
}],<br/>calculatedFieldSettings: [{<br/>name: 'Total',<br/>formula:  
    ""Sum(Amount)"+"Sum(Sold)"" }]<br/>};<br/>}}|
```

Paging

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

— — — — —

Paging property: enablePaging <ej-pivotgrid id="PivotGrid1" [enablePaging]="true"></ej-pivotgrid> Not Applicable
--

```
|Virtual scrolling|property: enableVirtualScrolling<br/><br/><ej-pivotgrid id="PivotGrid1"
[enableVirtualScrolling]="true">|property: enableVirtualization<br/><br/><ejs-pivotview
#pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings
enableVirtualization='true'></ejs-pivotview>|
```

Conditional Formatting

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

— — — — —

```
[Show/hide conditional formatting] | property: enableConditionalFormatting<br/><br/><ej-pivotgrid
[enableConditionalFormatting]="true"></ej-pivotgrid> | property:
allowConditionalFormatting<br/><br/><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings allowConditionalFormatting='true'></ejs-pivotview>
```

```
|Conditional formatting settings| property: conditionalFormatSettings<br/><br/><ej-pivotgrid
[conditionalFormatSettings]=“conditionalFormatSettings”></ej-pivotgrid><br/><br/>export class
PivotGridComponent {<br/>public conditionalFormatSettings;<br/>constructor()
{<br/>this.conditionalFormatSettings: [{<br/>name: “Format2”,<br/>style: {<br/>“color”:
“#000000”,<br/>“backgroundColor”: “#0000FF”,<br/>“bordercolor”:
“#000000”,<br/>“borderstyle”: “Dashed”,<br/>“borderwidth”: “5”,<br/>“fontsize”:
“12”,<br/>“fontstyle”: “Algerian” },<br/>condition:
ej.PivotGrid.ConditionalOptions.LessThan,<br/>value: “200”,<br/>measures: “Amount,Quantity”
}],<br/>}}| property: conditionalFormatSettings<br/><br/><ej-pivotview #pivotview id=‘PivotView’
[dataSourceSettings]=dataSourceSettings></ej-pivotview><br/><br/>export class
AppComponent implements OnInit {<br/>public dataSourceSettings:
IDataOptions;<br/>ngOnInit(): void {<br/>this.dataSourceSettings =
{<br/>conditionalFormatSettings: [{<br/>measure: ‘In Stock’,<br/>value1: 5000,<br/>conditions:
‘LessThan’,<br/>style: {<br/>backgroundColor: ‘#80cbc4’,<br/>color: ‘black’,<br/>fontFamily:
‘Tahoma’,<br/>fontSize: ‘12px’ }<br/>}}];<br/>}}|
```

Exporting

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

— — — — —

| Excel Export | Not Applicable | **property:** allowExcelExport

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings allowExcelExport='true'></ejs-pivotview>|

| Pdf Export | Not Applicable | **property:** allowPdfExport

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings allowPdfExport='true'></ejs-pivotview>|

Grid Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Set width for pivot table | Not Applicable | **property:** width

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings width='100%'></ejs-pivotview>|

| Set height for pivot table | Not Applicable | **property:** height

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings height='500'></ejs-pivotview>|

| Set row height for pivot table | Not Applicable | **property:** rowHeight

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
rowHeight: 60;
} as GridSettings;
}}|

| Set column width for pivot table | Not Applicable | **property:** columnWidth

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
columnWidth: 120;
} as GridSettings;
}}|

| Drag and drop column headers in pivot table | Not Applicable | **property:** allowReordering

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
allowReordering: true;
} as GridSettings;
}}|

| Resizing the column headers in pivot table | **property:** enableColumnResizing

<ej-pivotgrid [enableColumnResizing]="true"></ej-pivotgrid>| **property:** allowResizing

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
allowResizing: true;
} as GridSettings;
}}|

| Wrap the cell content in pivot table | Not Applicable | **property:** allowTextWrap

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
allowTextWrap: true;
} as GridSettings;
}}|

| Display cell border in pivot table | Not Applicable | **property:** gridLines

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
gridLines: 'Vertical';
} as GridSettings;
}}|

| Cell selection | **property:** enableCellSelection

<ej-pivotgrid [enableCellSelection]="true"></ej-pivotgrid> | **property:** allowSelection

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
allowSelection: true,
selectionSettings: {
cellSelectionMode: 'Box',
type: 'Multiple',
mode: 'Cell'}
} as GridSettings;
}}|

| Display overflow cell content in pivot table | Not Applicable | **property:** clipMode

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'></ejs-pivotview>

export class AppComponent implements OnInit {
public gridSettings: GridSettings;
ngOnInit(): void {
this.gridSettings = {
clipMode: 'Clip';
} as GridSettings;
}}|

| Cell Editing | **property:** enableCellEditing

<ej-pivotgrid [enableCellEditing]="true"></ej-pivotgrid> | Not Applicable |

| Cell double click | **property:** enableCellDoubleClick

<ej-pivotgrid [enableCellDoubleClick]="true"></ej-pivotgrid> | Not Applicable |

| Cell context | **property:** enableCellContext

<ej-pivotgrid [enableCellContext]="true"></ej-pivotgrid> | Not Applicable |

Accessibility

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Localization | **property:** locale

<ej-pivotgrid id="PivotGrid1" locale="fr-FR"></ej-pivotgrid> | **property:** locale

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings locale='de-DE'></ejs-pivotview> |

| Right to left | **property:** enableRTL

<ej-pivotgrid [enableRTL]="true"></ej-pivotgrid> | **property:** enableRtl

<ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings enableRtl=true></ejs-pivotview> |

Common

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Adding custom class to wrapper element | **property:** cssClass

<ej-pivotgrid [cssClass]="custom-class"></ej-pivotgrid> | **property:** cssClass

<ejs-pivotview #pivotview

```
id='PivotView' [dataSourceSettings]=dataSourceSettings cssClass='custom-class'></ejs-
pivotview>|
```

```
| Event triggers when control initializing | event: load<br/><br/><ej-pivotgrid
[dataSource]="dataSource" (load)="onLoad($event)"></ej-pivotgrid><br/><br/>export class
PivotGridComponent {<br/>onLoad(args) {}<br/>} | event: load<br/><br/><ejs-pivotview
#pivotview id='PivotView' (load)='onLoad($event)'></ejs-pivotview><br/><br/>export class
AppComponent {<br/>onLoad(): void {}<br/>}|
```

```
| Event Triggers before the pivot engine starts | event: beforePivotEnginePopulate<br/><br/><ej-
pivotgrid [dataSource]="dataSource"
(beforePivotEnginePopulate)="onbeforePivotEnginePopulate($event)"></ej-
pivotgrid><br/><br/>export class PivotGridComponent {<br/>onbeforePivotEnginePopulate(args)
{}<br/>} | event: enginePopulating<br/><br/><ejs-pivotview #pivotview id='PivotView'
(load)='onLoad($event)'></ejs-pivotview><br/><br/>export class AppComponent
{<br/>onLoad(): void {}<br/>}|
```

```
| Event Triggers after the pivot engine populated | event: afterPivotEnginePopulate<br/><br/><ej-
pivotgrid [dataSource]="dataSource"
(afterPivotEnginePopulate)="onafterPivotEnginePopulate($event)"></ej-
pivotgrid><br/><br/>export class PivotGridComponent {<br/>onafterPivotEnginePopulate(args)
{}<br/>} | event: enginePopulated<br/><br/><ejs-pivotview #pivotview id='PivotView'
(load)='onLoad($event)'></ejs-pivotview><br/><br/>export class AppComponent
{<br/>onLoad(): void {}<br/>}|
```

```
| Event Triggers after the control populated with data source | event: renderSuccess<br/><br/><ej-
pivotgrid [dataSource]="dataSource" (renderSuccess)="onrenderSuccess($event)"></ej-
pivotgrid><br/><br/>export class PivotGridComponent {<br/>onrenderSuccess(args)
{}<br/>} | event: dataBound<br/><br/><ejs-pivotview #pivotview id='PivotView'
(dataBound)='ondataBound($event)'></ejs-pivotview><br/><br/>export class AppComponent
{<br/>ondataBound(): void {}<br/>}|
```

```
| Event Triggers after the control created | Not Applicable | event: created<br/><br/><ejs-pivotview
#pivotview id='PivotView' (created)='oncreated($event)'></ejs-pivotview><br/><br/>export
class AppComponent {<br/>oncreated(): void {}<br/>}|
```

```
| Event Triggers when destroy the control | Not Applicable | event: destroyed<br/><br/><ejs-pivotview
#pivotview id='PivotView' (destroyed)='ondestroyed($event)'></ejs-pivotview><br/><br/>export
class AppComponent {<br/>ondestroyed(): void {}<br/>}|
```

```
| Event Triggers the cell clicked in pivot table widget | event: cellClick<br/><br/><ej-pivotgrid
[dataSource]="dataSource" (cellClick)="oncellClick($event)"></ej-pivotgrid><br/><br/>export
class PivotGridComponent {<br/>oncellClick(args) {}<br/>} | event: cellClick<br/><br/><ejs-
pivotview #pivotview id='PivotView' (cellClick)='oncellClick($event)'></ejs-
pivotview><br/><br/>export class AppComponent {<br/>oncellClick(): void {}<br/>}|
```

```
| Keeping the model values in cookies | Not Applicable | property: enablePersistence<br/><br/><ejs-
pivotview #pivotview id='PivotView' enablePersistence='true'></ejs-pivotview>|
```

How To

<!-- markdownlint-disable MD009 -->

Switching older themes style in Angular Pivotview component

From Volume 1, 2020 onwards Syncfusion has revised the theming and layout of the Pivot Table. So, to inherit the older theme style and layout please do the necessary changes in CSS and pivot table height.

CSS Selectors

In current theme, the cells can be differentiated by their background colors. To avoid it, you need to override its background colors via simple CSS coding within your application. The below CSS selectors allow to achieve the same for material, fabric, bootstrap and bootstrap v4 themes.

```
`html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Codes here... -->
<style>
.e-pivotview .e-rowsheader,
.e-pivotview .e-columnsheader,
.e-pivotview .e-gtot,
.e-pivotview .e-content,
.e-pivotview .e-gridheader,
.e-pivotview .e-headercell {
background-color:#fff !important;
}
</style>
</head>
<body>
</body>
</html>
`
```

Meanwhile for high contrast theme, we need to set the following CSS.

```
`html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- Codes here... -->
```

```

<style>
.e-pivotview .e-rowsheader,
.e-pivotview .e-columnsheader,
.e-pivotview .e-gtot,
.e-pivotview .e-content,
.e-pivotview .e-gridheader,
.e-pivotview .e-headercell {
background-color:#000 !important;
}
</style>
</head>
<body>
</body>
</html>
`

```

Adjusting Row Height

In current theme, to make the component compact we have reduced the height of each pivot table rows. But user can reset the height of the pivot table using the [rowHeight](#) property in [gridSettings](#). In older theme, the property was set to 36 pixels for desktop layout and 48 pixels for mobile layout. So reset the [rowHeight](#) accordingly to visualize the older theme style.

In the below code sample, we replicate the older theme style.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule
],
standalone: true,
selector: 'app-container',
// specifies the template string for the pivot table component
template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
[gridSettings]='gridSettings' width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {

```

```

public width?: string;
public dataSourceSettings?: IDataOptions;
public gridSettings?: GridSettings;
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.gridSettings = {
        rowHeight: 36
    } as GridSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize number date and time values in Angular Pivotview component

You can format the number, date, and time values for each field using **formatSettings** option under **dataSourceSettings**. It can be configured through code behind, during initial rendering.

Number formatting

For numbers, the formatting settings required to apply through code behind are:

- **name**: It allows to set the field name.
- **format**: It allows to set the format of the respective field.

Also, you can customize the applied number format by setting the [NumberFormatOptions](#) options in **formatSettings** itself.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
    imports: [

```



```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
      this.width = '100%';
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        formatSettings: [{ name: 'Amount', format: 'C2', useGrouping:
false,
                        minimumSignificantDigits: 1, maximumSignificantDigits: 3
}],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        filters: []
      };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Date and Time formatting

For date and time, the formatting settings required to apply through code behind are:

- **name**: It allows to set the field name.
- **format**: It allows to set the format of the respective field.
- **type**: It allows to set the type of format to be used for the respective field.

Also, you can customize the applied date format by setting [DateFormatOptions](#) options in **formatSettings** itself.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings width=width></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public width?: string;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      formatSettings: [{ name: 'Year', format: 'dd/MM/yyyy-hh:mm',
type: 'date' }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Limitations of date formatting

As per Firefox and Edge browsers standards, most of the date and time formats used in data source aren't supported. For example: Apr-2000, Apr-01-2000, 01-03-2000, 2000-Apr-01 etc... are not supported. Meanwhile [ISO formats](#) will be supported across all browsers.

Customize the icons for pivot grid in Angular Pivotview component

You can customize the pivot button icons in the pivot table by overriding the class **.pivot-button** with a custom property content as mentioned below.

`typescript

```
PivotView_PivotFieldList .e-icons.e-toggle-field-list::before {
content: '\e337';
}
`
```

In the below sample, pivot table is rendered with a customized pivot button icons.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
// specifies the template string for the pivot table component
template: `<div><ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [width]=width
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent {
    public dataSourceSettings?: IDataOptions;
    public width?: string;
    ngOnInit(): void {
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
        this.width = "100%";
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Configure data grid options on editing mode in Angular Pivotview component

You can access the data grid options such as sort, group, filter on editing mode using the **beginDrillThrough** event in the pivot table. The event occurs in every value cell on double click and provides the data grid information before display the drill through grid pop-up.

Grid features are segregated into individual feature-wise modules. For example, to use sorting feature, you should inject **Sort** using the **Grid.Inject(Sort)** section.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, CellEditSettings,
BeginDrillThroughEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { Grid, Sort, Filter, Group } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [editSettings]=editSettings
width=width (beginDrillThrough)='beginDrillThrough($event)'></ejs-
pivotview>`
})
export class AppComponent {
  public width?: string;
  public editSettings?: CellEditSettings;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
      values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
      filters: []
    };
    this.editSettings = {
```

```

        allowAdding: true, allowDeleting: true, allowEditing: true,
mode: 'Normal'
    } as CellEditSettings;
}
beginDrillThrough(args: BeginDrillThroughEventArgs) {
    if (args.gridObj) {
        Grid.Inject(Sort, Filter, Group);
        let gridObj: Grid = args.gridObj;
        gridObj.allowGrouping = true;
        gridObj.allowSorting = true;
        gridObj.allowFiltering = true;
        gridObj.filterSettings = { type: 'CheckBox' };
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Refresh the field list in Angular Pivotview component

You can refresh pivot table and field list with new data source dynamically.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { PivotViewComponent, IDataOptions, IDataset, FieldListService,
PivotView, DataSourceSettings, PivotEngine } from '@syncfusion/ej2-angular-
pivotview';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div class="col-md-8"><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings
  showFieldList="true" width=width height=height></ejs-
pivotview></div><div class="col-md-2">
  <button ej-button id='refresh'
(click)='applyData($event)'>Refresh</button></div>`
})
export class AppComponent implements OnInit {
  public pivotData?: IDataset[];

```

```

public dataSourceSettings?: IDataOptions;
public button?: Button;
public width?: string;
public height?: string;
@ViewChild('pivotview',{static: false})
public pivotGridObj?: PivotViewComponent;
applyData(e: Event): void {
    debugger;
    ((this.pivotGridObj as PivotViewComponent).engineModule as
PivotEngine).fieldList = {};
    ((this.pivotGridObj as PivotViewComponent).dataSourceSettings as
DataSourceSettings ).dataSource = [
        { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Year': 'FY
2015' },
        { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Year': 'FY
2015' },
        { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Year': 'FY
2015' },
        { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Year': 'FY
2015' },
        { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Year': 'FY
2016' }
    ] as IDataset[];
};
ngOnInit(): void {
    this.pivotData = [
        { 'Sold': 31, 'Amount': 52824, 'Country': 'France', 'Products':
'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q1' },
        { 'Sold': 51, 'Amount': 86904, 'Country': 'France', 'Products':
'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q2' },
        { 'Sold': 90, 'Amount': 153360, 'Country': 'France', 'Products':
'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q3' },
        { 'Sold': 25, 'Amount': 42600, 'Country': 'France', 'Products':
'Mountain Bikes', 'Year': 'FY 2015', 'Quarter': 'Q4' },
        { 'Sold': 27, 'Amount': 46008, 'Country': 'France', 'Products':
'Mountain Bikes', 'Year': 'FY 2016', 'Quarter': 'Q1' }
    ];
    this.dataSourceSettings = {
        dataSource: this.pivotData,
        expandAll: false,
        columns: [{ name: 'Year', caption: 'Production Year' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.width = '100%';
    this.height = '350';
    this.button = new Button({
        isPrimary: true,
    });
    this.button.appendTo('#refresh');
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Hide empty headers in Angular Pivotview component

If the raw data for a particular field is not defined, it will be shown as 'Undefined' in the pivot table headers. You can hide those headers by setting the [showHeaderWhenEmpty](#) property to **false** in the pivot table.

For example, if the raw data for the field 'Country' is defined as “United Kingdom” and “State” is not defined means, it will be shown as “United Kingdom >> Undefined” in the header section. Here, you can hide those 'Undefined' header using the [showHeaderWhenEmpty](#) property.

By default, this property is set as **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService } from
'@syncfusion/ej2-angular-pivotview';
import { pivotNullData } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='350' [dataSourceSettings]=dataSourceSettings
showFieldList='true' width=width></ejs-pivotview></div>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: pivotNullData as IDataset[],
      expandAll: false,
      rows: [{ name: 'Country' }, { name: 'State' }],
      columns: [{ name: 'Product', showNoDataItems: true }, { name:
'Date' }],
      values: [{ name: 'Amount' }, { name: 'Quantity' }],
      showHeaderWhenEmpty: false
    };
  }
}
```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customizing loading indicator in Angular Pivotview component

You can customize the appearance of the loading indicator in the pivot table by using the [spinnerTemplate](#) property. This property accepts an HTML string which can be used for appearance customization.

You can also disable the loading indicator by setting [spinnerTemplate](#) to empty string.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { DataManager, WebApiAdaptor, Query, ReturnOption } from
 '@syncfusion/ej2-data';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings width=width spinnerTemplate="<i
class='fa fa-cog fa-spin fa-3x fa-fw'></i>"></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public data?: DataManager;
  public width?: string;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://bi.syncfusion.com/northwindservice/api/orders',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.width = '100%';
    this.dataSourceSettings = {
      dataSource: this.data,
      expandAll: true,
      filters: [],
```



```

        columns: [{ name: 'ProductName', caption: 'Product Name' }],
        rows: [{ name: 'ShipCountry', caption: 'Ship Country' }, { name:
'ShipCity', caption: 'Ship City' }],
        formatSettings: [{ name: 'UnitPrice', format: 'C0' }],
        values: [{ name: 'Quantity' }, { name: 'UnitPrice', caption:
'Unit Price' }]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing the pivotview component minimum height in Angular Pivotview component

The `minHeight` property allows you to change the minimum height for the pivot table control. For the pivot table control, the default minimum height is **300px**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView } from '@syncfusion/ej2-angular-
pivotview';
import { Pivot_Data } from './datasource';
import { Button } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div style="height: 480px;"><ejs-pivotview #pivotview
id='PivotView' height='200' [dataSourceSettings]=dataSourceSettings
width=width (load)="load($event)"></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public dataSourceSettings?: IDataOptions;
  public button?: Button;
  public width?: string;
  @ViewChild('pivotview', {static: false})
  public pivotGridObj?: PivotView;
  load(args: any): void {
    if(this.pivotGridObj) {
      this.pivotGridObj.minHeight = 200;
    }
  }
}

```

```

ngOnInit(): void {
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        drilledMembers: [{ name: 'Country', items: ['France'] }],
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }, { name: 'Total', caption: 'Total
Amount', type: 'CalculatedField' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }, { name:
'Total', format: 'C2' }],
        filters: [],
        calculatedFieldSettings: [{ name: 'Total', formula:
'"Sum(Amount)"+"Sum(Sold)"' }]
    };
    this.width = '100%';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Chart based on pivot table selection in Angular Pivotview component

The cell selection support is enabled using the [allowSelection](#) property and its type and mode are configured using the [selectionSettings](#) property. The [cellSelected](#) event gets fired on every selection operation performed in the pivot table. This event returns the selected cell informations, like row header name, column header name, measure name, and value. Based on this information, the [chart](#) control will be plotted.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import { IDataOptions, PivotView, IAxisSet, IFieldOptions,
PivotViewComponent, FieldListService, PivotCellSelectedEventArgs,
CellSelectedObject, DataSourceSettings, IDataset } from '@syncfusion/ej2-
angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Chart, Category, Legend, Tooltip, ColumnSeries, LineSeries,
SeriesModel } from '@syncfusion/ej2-charts';
import { renewableEnergy } from './datasource';
@Component({
  imports: [

```

```

        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    // specifies the template string for the pivot table component
    template: `<div><ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings showFieldList='true' width=width
height=height [gridSettings]=gridSettings
(cellSelected)="cellSelected($event)" (dataBound)="dataBound($event)"></ejs-
pivotview><div><br/><div id="Chart"></div>`
    })
    export class AppComponent implements OnInit {
        public width?: string;
        public height?: number;
        public dataSourceSettings?: IDataOptions;
        public gridSettings?: GridSettings;
        public onInit: boolean = true;
        public measureList: { [key: string]: string } = {};
        public chart?: Chart;
        public selectedCells?: CellSelectedObject[];
        public chartSeries?: SeriesModel[];
        @ViewChild('pivotview', {static: false})
        public pivotObj?: PivotViewComponent;
        frameChartSeries(): SeriesModel[] {
            let columnGroupObject: { [key: string]: { x: string, y: number }[] }
            = {};
            for (let cell of this.selectedCells!) {
                if (cell.measure !== '') {
                    let columnSeries = ((this.pivotObj?.dataSourceSettings as
DataSourceSettings).values.length > 1 && this.measureList[cell.measure]) ?
                    (cell.columnHeaders.toString() + ' ~ ' +
this.measureList[cell.measure]) : cell.columnHeaders.toString();
                    if (columnGroupObject[columnSeries]) {
                        columnGroupObject[columnSeries].push({ x: cell.rowHeaders == ''
? 'Grand Total' : cell.rowHeaders.toString(), y: Number(cell.value) });
                    } else {
                        columnGroupObject[columnSeries] = [{ x: cell.rowHeaders == '' ?
'Grand Total' : cell.rowHeaders.toString(), y: Number(cell.value) }];
                    }
                }
            }
            let columnKeys: string[] = Object.keys(columnGroupObject);
            let chartSeries: SeriesModel[] = [];
            for (let key of columnKeys) {
                chartSeries.push({
                    dataSource: columnGroupObject[key],
                    xName: 'x',
                    yName: 'y',
                    type: 'Column',
                    name: key
                });
            }
            return chartSeries;
        }
        chartUpdate(): void {

```

```

        if (this.onInit) {
            this.onInit = false;
            Chart.Inject(ColumnSeries, LineSeries, Legend, Tooltip,
Category);
            this.chart = new Chart({
                title: 'Sales Analysis',
                legendSettings: {
                    visible: true
                },
                tooltip: {
                    enable: true
                },
                primaryYAxis: {
                    title: (this.pivotObj?.dataSourceSettings as
DataSourceSettings).values.map(function (args: IFieldOptions) { return
args.caption || args.name }).join(' ~ '),
                },
                primaryXAxis: {
                    valueType: 'Category',
                    title: (this.pivotObj?.dataSourceSettings as
DataSourceSettings).rows.map(function (args: IFieldOptions) { return
args.caption || args.name }).join(' ~ '),
                    labelIntersectAction: 'Rotate45'
                },
                series: this.chartSeries,
            }, '#Chart');
        } else {
            (this.chart as Chart).series = this.chartSeries as
SeriesModel[];
            (this.chart as Chart).primaryXAxis.title =
(this.pivotObj?.dataSourceSettings as DataSourceSettings).rows.map(function
(args: IFieldOptions) { return args.caption || args.name }).join(' ~ ');
            (this.chart as Chart).primaryYAxis.title =
(this.pivotObj?.dataSourceSettings as
DataSourceSettings).values.map(function (args: IFieldOptions) { return
args.caption || args.name }).join(' ~ ');
            this.chart?.refresh();
        }
    }
    dataBound(args: any): void {
        if(this.onInit) {
            for (let value of (this.pivotObj?.dataSourceSettings as
DataSourceSettings).values) {
                this.measureList[(value as any).name] = (value.caption ||
value.name) as string;
            }
            this.pivotObj?.grid.selectionModule.selectCellsByRange(
                { cellIndex: 1, rowIndex: 1 },
                { cellIndex: 3, rowIndex: 3 }
            );
        }
    }
    cellSelected(args: PivotCellSelectedEventArgs): void {
        this.selectedCells = args.selectedCellsInfo;
        if (this.selectedCells && this.selectedCells.length > 0) {
            this.chartSeries = this.frameChartSeries();
            this.chartUpdate();
        }
    }

```

```

    }
  }
  ngOnInit(): void {
    this.width = "100%";
    this.height = 350;
    this.gridSettings = {
      columnWidth: 120,
      allowSelection: true,
      selectionSettings: {
        mode: 'Cell',
        type: 'Multiple',
        cellSelectionMode: 'Box',
      }
    } as GridSettings;
    this.dataSourceSettings = {
      dataSource: renewableEnergy as IDataset[],
      expandAll: true,
      enableSorting: true,
      drilledMembers: [{ name: 'Year', items: ['FY 2015', 'FY 2017',
'FY 2018'] }],
      formatSettings: [{ name: 'ProCost', format: 'C0' }],
      rows: [
        { name: 'Year', caption: 'Production Year' }
      ],
      columns: [
        { name: 'EnerType', caption: 'Energy Type' },
        { name: 'EneSource', caption: 'Energy Source' }
      ],
      values: [
        { name: 'ProCost', caption: 'Revenue Growth' }
      ],
      filters: []
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Drill through grid cell edit type in Angular Pivotview component

Using the [drillThrough](#) event in the pivot table, you can define the edit type of a particular column in the grid present inside the drill-through dialog. To do so, check the column name in the [drillThrough](#) event and then specify the edit type of that column using the [gridColumns.editType](#) event argument.

The `[gridColumns.editType]` property must be set based on the column's data type. For example, the string data type will not be applicable for the numeric text box edit type.

- [NumericTextBox](#) control for integer, double, and decimal data types.
- [TextBox](#) control for string data type.
- [DropDownList](#) control to show all unique values related to that field.

- [CheckBox](#) control for boolean data type.
- [DatePicker](#) control for date data type.
- [DateTimePicker](#) control for date time data type.

In the below example, the data type of the **Country** column is set to **DropDownList**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component } from '@angular/core';
import { IDataOptions, IDataset, PivotView, DrillThroughService,
DrillThroughEventArgs } from '@syncfusion/ej2-angular-pivotview';
import { CellEditSettings } from '@syncfusion/ej2-pivotview/src/pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [DrillThroughService],
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings [editSettings]='editSettings'
width=width (drillThrough)='drillThrough($event)'></ejs-pivotview>`
})
export class AppComponent {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  public editSettings?: CellEditSettings;
  drillThrough(args: DrillThroughEventArgs | any) {
    for (var i = 0; i < args.gridColumns.length; i++) {
      if (args.gridColumns[i].field === 'Country') {
        args.gridColumns[i].editType = 'dropdownedit';
        //args.gridColumns[i].editType = 'numericedit';
        //args.gridColumns[i].editType = 'textedit';
        //args.gridColumns[i].editType = 'booleanedit';
        //args.gridColumns[i].editType = 'datepickeredit';
        //args.gridColumns[i].editType = 'datetimepickeredit';
      }
    }
  }
  ngOnInit(): void {
    this.width = "100%";
    this.dataSourceSettings = {
      dataSource: Pivot_Data as IDataset[],
      expandAll: false,
      enableSorting: true,
      drilledMembers: [{ name: 'Country', items: ['France'] }],
      columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
```

```

        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
    this.editSettings = {
        allowEditing: true, allowAdding: true, allowDeleting: true,
mode: 'Normal', allowCommandColumns: false,
        allowEditOnDbClick: true, showConfirmDialog: true,
showDeleteConfirmDialog: false
    } as CellEditSettings;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show field list when pivot table empty in Angular Pivotview component

When there are no fields in a pivot table's row, column, value, and filter axes, a field list can still be displayed. To do so, use the [dataBound](#) event and call the `onShowFieldList` method as shown below.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { IDataOptions, IDataset, PivotView, FieldListService,
DataSourceSettings } from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView' height='350'
[dataSourceSettings]=dataSourceSettings
width=width (dataBound)='ondataBound()' showFieldList='true'></ejs-
pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public dataSourceSettings?: IDataOptions;
  @ViewChild('pivotview',{static: false})

```

```

public pivotGridObj?: PivotView;
ondataBound(): void {
    if (this.pivotGridObj && (this.pivotGridObj?.dataSourceSettings as
DataSourceSettings).values.length === 0) {
        (this.pivotGridObj?.pivotFieldListModule.dialogRenderer as
any).onShowFieldList();
    }
}
ngOnInit(): void {
    this.width = '100%';
    this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Apply custom style to pivot cells in Angular Pivotview component

The [queryCellInfo](#) event in [gridSettings](#) can be used to apply custom style to row and value cells, and the [headerCellInfo](#) event in [gridSettings](#) can be used to apply custom styles to column cells.

In the following example, a custom style has been applied to the column header **“Sold Amount”** under **“FY 2016”** via the [headerCellInfo](#) event and to the row header **“Germany”** and its aggregated value via the [queryCellInfo](#) event by adding the **“e-custom”** class to the cell element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions } from
'@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Grid } from '@syncfusion/ej2-grids';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component

```



```

    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings [gridSettings]='gridSettings'
    width=width height=height showFieldList='true'
(enginePopulated)='enginePopulated($event)'></ejs-pivotview>`
  })
  export class AppComponent implements OnInit {
    public width?: string;
    public height?: number;
    public dataSourceSettings?: IDataOptions;
    public gridSettings?: GridSettings;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    queryCellInfo(args: any): void {
      let colIndex: number = Number(args.cell.getAttribute('data-colindex'));
      let cells: any = args.data[colIndex] ? args.data[colIndex] : {};
      // Here by using 'actualText' option, a custom class can be added to the
specific row header and its value to apply custom style.
      if (cells.actualText === 'Germany') {
        args.cell.classList.add('e-custom');
      } else if (cells.actualText === 'Amount' &&
        cells.columnHeaders === 'FY 2016' && cells.rowHeaders === 'Germany') {
        args.cell.classList.add('e-custom');
      }
    }
    headerCellInfo(args: any): void {
      let customAttributes: any = args.cell.column.customAttributes;
      // Here custom class can be added to the specific column header by using
unique level name, to apply custom style.
      if (args.node.classList.contains('e-columnsheader') && customAttributes
&&
        customAttributes.cell.valueSort.levelName === 'FY 2016.Sold Amount') {
        args.node.classList.add('e-custom');
      }
    }
    enginePopulated(args: any): void {
      ((this.pivotGridObj as PivotView).grid as Grid).queryCellInfo =
this.queryCellInfo.bind(this);
      ((this.pivotGridObj as PivotView).grid as Grid).headerCellInfo =
this.headerCellInfo.bind(this);
    }
    ngOnInit(): void {
      this.width = "100%";
      this.height = 350;
      this.dataSourceSettings = {
        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
      };
      this.gridSettings = {
        columnWidth: 140,

```

```

    } as GridSettings;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Note: The **dot(.)** character in **FY 2016.Sold Amount** is used by default to identify the header levels in the pivot table's row and column. It can be changed by setting the [headerDelimiter](#) in the [valueSortSettings](#) property to any other delimiter instead of the default separator.

Show tooltip for row and column headers in Angular Pivotview component

You can create and display the tooltip for each row and column header(s) in the pivot table by using an external tooltip component via the [dataBound](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions, IAxisSet }
from '@syncfusion/ej2-angular-pivotview';
import { Tooltip } from '@syncfusion/ej2-popups';
import { Pivot_Data } from './datasource';
let headerTooltip: Tooltip;
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule

  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings (dataBound)="dataBound($event)"
width=width height=height showFieldList='true'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public height?: number;
  public dataSourceSettings?: IDataOptions;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj?: any;
  dataBound(args: any): void {
    if (!headerTooltip) {
      headerTooltip = new Tooltip({

```

```

        target: 'td.e-rowsheader,th.e-columnsheader', beforeRender:
this.beforeRender.bind(this)
    });
    headerTooltip.appendTo(this.pivotGridObj?.element);
}
}
beforeRender(args: any) {
    if (args.target.parentElement.querySelector('.e-rowsheader')) {
        // Here you can set custom content for row header(s) tooltip from its
        cell information.
        let index: number =
Number(args.target.getAttributeNode('index').value);
        let colIndex: number = Number(args.target.getAttributeNode('data-
colindex').value);
        let cell: IAxisSet =
this.pivotGridObj?.engineModule.pivotValues[index][colIndex];
        let valueText: any = cell.valueSort ? cell.valueSort : '';
        if (cell.formattedText !== 'Grand Total') {
            headerTooltip.content =
                '<div>' +
                'FieldName: ' +
                valueText.axis +
                '</br>' +
                'Text: ' +
                cell.formattedText +
                '</div>';
        } else {
            headerTooltip.content =
                '<div>' +
                'FieldName: ' +
                valueText.uniqueName +
                '</br>' +
                'Text: ' +
                cell.formattedText +
                '</div>';
        }
    } else {
        // Here you can set custom content for column header(s) tooltip from
        its cell information.
        if (args.target.querySelector('.e-cellvalue')) {
            headerTooltip.content = args.target.querySelector('.e-
cellvalue').innerText;
        } else if (args.target.querySelector('.e-headertext')) {
            headerTooltip.content = args.target.querySelector('.e-
headertext').innerText;
        } else if (args.target.querySelector('.e-stackedheadercelldiv')) {
            headerTooltip.content = args.target.querySelector('.e-
stackedheadercelldiv').innerText;
        } else {
            headerTooltip.content = '';
        }
    }
}
}
ngOnInit(): void {
    this.width = "100%";
    this.height = 350;
    this.dataSourceSettings = {

```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hide specific columns in Angular Pivotview component

By using the [columnRender](#) event in the [gridSettings](#), you can hide specific column(s) in the pivot table. In the example below, the “Units Sold” column under “FY 2016” is hidden by setting its **visible** property to **false** via the [columnRender](#) event.

Note: The **dot(.)** character in **FY 2016.Units Sold** is used by default to identify the header levels in the pivot table's row and column. It can be changed by setting the [headerDelimiter](#) in the [valueSortSettings](#) property to any other delimiter instead of the default separator.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions } from
'@syncfusion/ej2-angular-pivotview';
import { GridSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/gridsettings';
import { Observable } from 'rxjs';
import { Pivot_Data } from './datasource';
@Component({
  imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  // specifies the template string for the pivot table component
  template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings [gridSettings]=gridSettings

```

```

        width=width height=height showFieldList='true'
        (columnRender)='columnRender($event)'></ejs-pivotview>`
    })
    export class AppComponent implements OnInit {
        columnRender($event: Event) {
        }
        public width?: string;
        public height?: number;
        public dataSourceSettings?: IDataOptions;
        public gridSettings?: GridSettings;
        public observable = new Observable();
        @ViewChild('pivotview', { static: false })
        public pivotGridObj?: PivotView;
        ngOnInit(): void {
            this.width = "100%";
            this.height = 350;
            this.gridSettings = {
                columnRender: this.observable.subscribe((args: any) => {
                    for (let i = 1; i < args.columns.length; i++) {
                        if (args.stackedColumns[i].customAttributes &&
                            args.stackedColumns[i].customAttributes.cell.valueSort.levelName
                            === 'FY 2016.Units Sold') {
                            args.stackedColumns[i].visible = false;
                        }
                    }
                }) as any
            } as GridSettings;
            this.dataSourceSettings = {
                dataSource: Pivot_Data as IDataset[],
                expandAll: false,
                enableSorting: true,
                columns: [{ name: 'Year', caption: 'Production Year' }, { name:
                'Quarter' }],
                values: [{ name: 'Sold', caption: 'Units Sold' }, { name: 'Amount',
                caption: 'Sold Amount' }],
                rows: [{ name: 'Country' }, { name: 'Products' }],
                formatSettings: [{ name: 'Amount', format: 'C0' }],
                filters: []
            };
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export table and chart into the same document using toolbar

Even if the [displayOption.view](#) property is set to **Both** in the pivot table, you can only export either the table or the chart to the PDF document based on the current value set in the [displayOption.primary](#) property. But, to export both the table and the chart to the same PDF document, use the [pdfExport](#) method during the [actionBegin](#) event invoke.

In the following example, the built-in export action can be restricted by setting the [args.cancel](#) option to **true** in the [actionBegin](#) event, and both the table and the chart can be exported by calling the [pdfExport](#) method and setting the `exportBothTableAndChart` argument to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
 '@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import {
    PivotView, FieldListService, IDataset, IDataOptions, PDFExportService,
    PivotActionBeginEventArgs, ToolbarItems, DisplayOption, ToolbarService,
    PivotChartService
} from '@syncfusion/ej2-angular-pivotview';
import { ChartSettings } from '@syncfusion/ej2-
pivotview/src/pivotview/model/chartsettings';
import { Pivot_Data } from './datasource';
@Component({
    imports: [
        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService, ToolbarService, PDFExportService,
    PivotChartService],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings allowPdfExport='true'
showFieldList='true' width='100%' [displayOption]='displayOption'
height='350' [toolbar]='toolbarOptions' showFieldList='true'
(actionBegin)='actionBegin($event)' showToolbar='true'
[chartSettings]='chartSettings'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    public toolbarOptions?: ToolbarItems[];
    public displayOption?: DisplayOption;
    public chartSettings?: ChartSettings;
    actionBegin(args: PivotActionBeginEventArgs): void {
        if (args.actionName == 'PDF export') {
            args.cancel = true;
            this.pivotGridObj.pdfExport({}, false, undefined, false, true);
        }
    }
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: any;
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['Grid', 'Chart', 'Export', 'FieldList'] as
        ToolbarItems[];
        this.chartSettings = { chartSeries: { type: 'Column' } } as
        ChartSettings;
        this.dataSourceSettings = {
```

```

        dataSource: Pivot_Data as IDataset[],
        expandAll: false,
        enableSorting: true,
        columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
        values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
        rows: [{ name: 'Country' }, { name: 'Products' }],
        formatSettings: [{ name: 'Amount', format: 'C0' }],
        filters: []
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD009 -->

Add custom aggregation type to the menu in Angular Pivotview component

By using the [dataBound](#) event, you can add your own custom aggregate type(s) to the pivot table's aggregate menu.

In the following example, we have added the aggregation types **CustomAggregateType 1** and **CustomAggregateType 2** to the aggregate menu. The calculation for those aggregated types can be done using the [aggregateCellInfo](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import { PivotView, FieldListService, IDataset, IDataOptions, AggregateTypes
} from '@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
import { L10n } from '@syncfusion/ej2-base';
let SummaryType: string[] = [
    'Sum',
    'Count',
    'DistinctCount',
    'Avg',
    'CustomAggregateType1',
    'CustomAggregateType2',
];
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],

```

```

standalone: true,
  selector: 'app-container',
  providers: [FieldListService],
  template: `<div class="col-md-8"> <ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings width=width
height=height
  (aggregateCellInfo)='aggregateCellInfo($event)' showFieldList='true'
  (dataBound)="dataBound($event)"></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
  public width?: string;
  public height?: number;
  public dataSourceSettings?: IDataOptions;
  @ViewChild('pivotview', { static: false })
  public pivotGridObj: any;
  aggregateCellInfo(args: any) {
    if (args.aggregateType === 'CustomAggregateType1') {
      args.value = args.value * 100;
    }
    if (args.aggregateType === 'CustomAggregateType2') {
      args.value = args.value / 100;
    }
  }
  dataBound(args: any): void {
    this.pivotGridObj.getAllSummaryType = function () {
      return SummaryType as AggregateTypes[];
    };
    this.pivotGridObj.pivotFieldListModule.aggregateTypes = SummaryType
as AggregateTypes[];
    this.pivotGridObj.pivotFieldListModule.getAllSummaryType = function
() {
      return SummaryType as AggregateTypes[];
    };
  }
  ngOnInit(): void {
    L10n.load({
      'en-US': {
        pivotview: {
          CustomAggregateType1: 'Custom Aggregate Type 1',
          CustomAggregateType2: 'Custom Aggregate Type 2',
        },
        pivotfieldlist: {
          CustomAggregateType1: 'Custom Aggregate Type 1',
          CustomAggregateType2: 'Custom Aggregate Type 2',
        }
      }
    });
    this.width = "100%";
    this.height = 350;
    this.dataSourceSettings = {
      expandAll: false,
      dataSource: Pivot_Data as IDataset[],
      columns: [{ name: 'Year' }, { name: 'Quarter' }],
      values: [{ name: 'Sold' }, { name: 'Amount' }],
      rows: [{ name: 'Country' }, { name: 'Products' }],
      formatSettings: [{ name: 'Amount', format: 'C0' }],
    };
  }
}

```



```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

<!-- markdownlint-disable MD009 -->

Convert complex JSON to flat JSON and assign it to the pivot table in Angular Pivotview component

By default, flat JSON can only bind to the pivot table. However, you can connect complex JSON to the pivot table by converting it to flat JSON via code-behind and binding it to the pivot table using the [dataSource](#) property in the [load](#) event.

In the following example, the **complexToFlatJson()** method is used to convert complex JSON to flat JSON and bind it to the pivot table using the [dataSource](#) property, then modifying the field names in the [rows](#) and [columns](#) based on the converted flat JSON under [dataSourceSettings](#) in the [load](#) event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, ViewChild, OnInit } from '@angular/core';
import {
    PivotView, FieldListService, IDataOptions, LoadEventArgs
} from '@syncfusion/ej2-angular-pivotview';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(false);
let parentProp: any = {};
let dataSource: Object[][];
@Component({
    imports: [
        PivotViewAllModule,
        PivotFieldListAllModule
    ],
    standalone: true,
    selector: 'app-container',
    providers: [FieldListService],
    // specifies the template string for the pivot table component
    template: `<ejs-pivotview #pivotview id='PivotView'
[dataSourceSettings]=dataSourceSettings
    showFieldList='true' width='100%' height='350' showFieldList='true'
(load)='load($event)'
    showToolbar='true'></ejs-pivotview>`
})
export class AppComponent implements OnInit {
    public dataSourceSettings?: IDataOptions;
    @ViewChild('pivotview', { static: false })
```

```

    public pivotGridObj?: PivotView;
    load(args: LoadEventArgs | any) {
        dataSource =
JSON.parse(JSON.stringify(args.dataSourceSettings.dataSource));
        args.dataSourceSettings.dataSource =
    this.complexToFlatJson(dataSource);
        let rows: any = [];
        for (let i: number = 0; i < args.dataSourceSettings.rows.length;
i++) {
            if (args.dataSourceSettings.rows[i].name in parentProp) {
                rows =
rows.concat(parentProp[args.dataSourceSettings.rows[i].name]);
            } else {
                rows.push(args.dataSourceSettings.rows[i]);
            }
        }
        args.dataSourceSettings.rows = rows;
        let columns: any = [];
        for (let i: number = 0; i < args.dataSourceSettings.columns.length;
i++) {
            if (args.dataSourceSettings.columns[i].name in parentProp) {
                columns = columns.concat(
                    parentProp[args.dataSourceSettings.columns[i].name]
                );
            } else {
                columns.push(args.dataSourceSettings.columns[i]);
            }
        }
        args.dataSourceSettings.columns = columns;
    }
    complexToFlatJson(data: Object[][][]) {
        let flatArray: any = [];
        let flatObject: any = {};
        for (let index = 0; index < data.length; index++) {
            for (let prop in data[index]) {
                let value: Object = data[index][prop];
                if (Array.isArray(value)) {
                    for (let i: number = 0; i < value.length; i++) {
                        let childProp: any = [];
                        for (let inProp in value[i]) {
                            flatObject[inProp] = value[i][inProp];
                            let object = {
                                name: inProp,
                            };
                            childProp.push(object);
                        }
                        parentProp[prop] = childProp;
                    }
                } else {
                    flatObject[prop] = value;
                }
            }
            flatArray.push(flatObject);
            flatObject = {};
        }
        return flatArray;
    }
}

```

```

data() {
  return [
    {
      CustomerID: 'VINET',
      Freight: 32.38,
      OrderDetails: [
        {
          OrderID: 10248,
          OrderDate: '1996-07-04T10:10:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShipName: 'Vins et alcools Chevalier',
          ShipAddress: '59 rue de l'Abbaye',
          ShipCity: 'Reims',
          ShipRegion: null,
          ShipCountry: 'France',
          ShippedDate: '1996-07-16T12:20:00.000Z',
        }
      ]
    },
    {
      CustomerID: 'GALED',
      Freight: 10.14,
      OrderDetails: [
        {
          OrderID: 10366,
          OrderDate: '1996-11-28T00:00:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShippedDate: '1996-12-30T00:00:00.000Z',
          ShipName: 'Galería del gastronómo',
          ShipAddress: 'Rambla de Cataluña, 23',
          ShipCity: 'Barcelona',
          ShipRegion: null,
          ShipCountry: 'Spain',
        }
      ]
    },
    {
      CustomerID: 'VAFFE',
      Freight: 13.55,
      OrderDetails: [
        {
          OrderID: 10367,
          OrderDate: '1996-12-02T00:00:00.000Z',
        }
      ],
      ShipDetails: [
        {
          ShippedDate: '1996-12-30T00:00:00.000Z',
          ShipName: 'Vaffeljernet',
          ShipAddress: 'Smagsloget 45',
          ShipCity: 'Århus',
        }
      ]
    }
  ]
}

```

```

                ShipRegion: null,
                ShipCountry: 'Denmark',
            }
        ]
    },
    {
        CustomerID: 'ERNSH',
        Freight: 101.95,
        OrderDetails: [
            {
                OrderID: 10368,
                OrderDate: '1996-11-29T00:00:00.000Z',
            }
        ],
        ShipDetails: [
            {
                ShippedDate: '1996-12-30T00:00:00.000Z',
                ShipName: 'Ernst Handel',
                ShipAddress: 'Kirchgasse 6',
                ShipCity: 'Graz',
                ShipRegion: null,
                ShipCountry: 'Austria',
            }
        ]
    },
    {
        CustomerID: 'SPLIR',
        Freight: 195.68,
        OrderDetails: [
            {
                OrderID: 10369,
                OrderDate: '1996-11-28T00:00:00.000Z',
            }
        ],
        ShipDetails: [
            {
                ShippedDate: '1996-12-30T00:00:00.000Z',
                ShipName: 'Split Rail Beer & Ale',
                ShipAddress: 'P.O. Box 555',
                ShipCity: 'Lander',
                ShipRegion: 'WY',
                ShipCountry: 'USA',
            }
        ]
    }
];

}
ngOnInit(): void {
    this.dataSourceSettings = {
        expandAll: true,
        enableSorting: true,
        dataSource: this.data() as any,
        columns: [{ name: 'OrderDetails' }],
        values: [{ name: 'Freight', caption: 'Units Sold' }],
        rows: [{ name: 'ShipDetails' }],
        valueSortSettings: { headerDelimiter: ' - ' },
        formatSettings: [{ name: 'Amount', format: 'C0' }]
    };
}

```

```

    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD009 -->

Load desired report from the report list as default in Angular Pivotview component

By default, the pivot table is displayed with the report bound at the code-behind. To load a desired report from the previously saved report collection during initial rendering, set the desired report name in the [dataBound](#) event, along with the additional report-based customization code shown below.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from '@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';
import {
  IDataOptions, PivotView, FieldListService, CalculatedFieldService,
  ToolbarService, ConditionalFormattingService, ToolbarItems,
  DisplayOption, IDataset
} from '@syncfusion/ej2-angular-pivotview';
import { getInstance, select } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { Pivot_Data } from './datasource';
let isInitial: boolean = true;
@Component({
  imports: [
    PivotViewAllModule,
    PivotFieldListAllModule
  ],
  standalone: true,
  selector: 'app-container',
  providers: [CalculatedFieldService, ToolbarService, ConditionalFormattingService, FieldListService],
  template: `<div><ejs-pivotview #pivotview id='PivotView' [dataSourceSettings]=dataSourceSettings allowExcelExport='true' allowConditionalFormatting='true' allowPdfExport='true' showToolbar='true' allowCalculatedField='true' showFieldList='true' width='100%' [displayOption]='displayOption' height='350' [toolbar]='toolbarOptions' (saveReport)='saveReport($event)' (loadReport)='loadReport($event)' (fetchReport)='fetchReport($event)' (renameReport)='renameReport($event)' (removeReport)='removeReport($event)'`
})

```

```

        (newReport)='newReport()' (dataBound)="dataBound()"
        (load)="load()"></ejs-pivotview></div>`
    })
    export class AppComponent implements OnInit {
        public dataSourceSettings?: IDataOptions;
        public toolbarOptions?: ToolbarItems[];
        public displayOption?: DisplayOption;
        @ViewChild('pivotview', { static: false })
        public pivotGridObj?: any;
        saveReport(args: any) {
            let reports = [];
            let isSaved: boolean = false;
            if (localStorage['pivotviewReports'] &&
                localStorage['pivotviewReports'] !== "") {
                reports = JSON.parse(localStorage['pivotviewReports']);
            }
            if (args.report && args.reportName && args.reportName !== '') {
                reports.map(function (item: any): any {
                    if (args.reportName === item.reportName) {
                        item.report = args.report; isSaved = true;
                    }
                });
                if (!isSaved) {
                    reports.push(args);
                }
                localStorage['pivotviewReports'] = JSON.stringify(reports);
            }
        }
        fetchReport(args: any) {
            let reportCollection: string[] = [];
            let reeportList: string[] = [];
            if (localStorage['pivotviewReports'] &&
                localStorage['pivotviewReports'] !== "") {
                reportCollection = JSON.parse(localStorage['pivotviewReports']);
            }
            reportCollection.map(function (item: any): void {
                reeportList.push(item.reportName);
            });
            args.reportName = reeportList;
        }
        loadReport(args: any) {
            let reportCollection: string[] = [];
            if (localStorage['pivotviewReports'] &&
                localStorage['pivotviewReports'] !== "") {
                reportCollection = JSON.parse(localStorage['pivotviewReports']);
            }
            reportCollection.map(function (item: any): void {
                if (args.reportName === item.reportName) {
                    args.report = item.report;
                }
            });
            if (args.report) {
                this.pivotGridObj.dataSourceSettings =
                    JSON.parse(args.report).dataSourceSettings;
            }
        }
        removeReport(args: any) {
            let reportCollection: any[] = [];

```

```

        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        for (let i: number = 0; i < reportCollection.length; i++) {
            if (reportCollection[i].reportName === args.reportName) {
                reportCollection.splice(i, 1);
            }
        }
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
        }
    }
    renameReport(args: any) {
        let reportCollection: string[] = [];
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            reportCollection = JSON.parse(localStorage['pivotviewReports']);
        }
        reportCollection.map(function (item: any): any { if (args.reportName
=== item.reportName) { item.reportName = args.rename; } });
        if (localStorage['pivotviewReports'] &&
localStorage['pivotviewReports'] !== "") {
            localStorage['pivotviewReports'] =
JSON.stringify(reportCollection);
        }
    }
    }
    dataBound() {
        // Set the default report name to load it in the pivot table during
initial rendering.
        if (this.pivotGridObj && isInitial) {
            isInitial = false;
            this.pivotGridObj.toolbarModule.action = 'Load';
            let reportList = getInstance(select('#' +
this.pivotGridObj.element.id + '_reportlist', this.pivotGridObj.element),
DropDownList);
            (reportList as DropDownList).value = 'Default report';
            this.loadReport({ reportName: 'Default report' });
        }
    }
    load(){
        // Save the desired report that needs to be loaded at initial
rendering here.
        let dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            columns: [{ name: 'Year' }],
            enableSorting: true,
            allowLabelFilter: true,
            values: [{ name: 'Sold', caption: 'Units Sold' }],
            allowValueFilter: true,
            formatSettings: [{ name: 'Sold', format: 'C0' }],
            rows: [{ name: 'Country' }],
        };
        let displayOption = { view: 'Both' };
        let gridSettings = {columnWidth: 100};

```

```

        let report = { dataSourceSettings: dataSourceSettings,
displayOption: displayOption, gridSettings: gridSettings };
        let reports = [
            {
                report: JSON.stringify(report),
                reportName: 'Default report',
            },
        ];
        localStorage['pivotviewReports'] = JSON.stringify(reports);
    }
    newReport() {
        this.pivotGridObj?.setProperties({ dataSourceSettings: { columns:
[], rows: [], values: [], filters: [] } }, false);
    }
    ngOnInit(): void {
        this.displayOption = { view: 'Both' } as DisplayOption;
        this.toolbarOptions = ['New', 'Save', 'SaveAs', 'Rename', 'Remove',
'Load',
        'Grid', 'Chart', 'Export', 'SubTotal', 'GrandTotal',
'ConditionalFormatting', 'FieldList'] as ToolbarItems[];
        this.dataSourceSettings = {
            dataSource: Pivot_Data as IDataset[],
            expandAll: false,
            enableSorting: true,
            drilledMembers: [{ name: 'Country', items: ['France'] }],
            columns: [{ name: 'Year', caption: 'Production Year' }, { name:
'Quarter' }],
            values: [{ name: 'Sold', caption: 'Units Sold' }, { name:
'Amount', caption: 'Sold Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }],
            formatSettings: [{ name: 'Amount', format: 'C0' }],
            filters: []
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Display string value to pivot table values

End user can display string value to the pivot table's value cell by using the [aggregateCellInfo](#) event.

In the following example, each cell value of the **Sold** field's actual value has been assigned from its combination data sets obtained from the [args.cellSets](#) in the [aggregateCellInfo](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PivotViewAllModule, PivotFieldListAllModule } from
'@syncfusion/ej2-angular-pivotview'
import { Component, OnInit, ViewChild } from '@angular/core';

```



```

import { PivotView, FieldListService, IDataset, IDataOptions } from
'@syncfusion/ej2-angular-pivotview';
import { Pivot_Data } from './datasource';
@Component({
imports: [

    PivotViewAllModule,
    PivotFieldListAllModule
],
standalone: true,
selector: 'app-container',
providers: [FieldListService],
template: `<div class="col-md-8"> <ejs-pivotview #pivotview
id='PivotView' [dataSourceSettings]=dataSourceSettings width=width
height=height
    (aggregateCellInfo)='aggregateCellInfo($event)'
showFieldList='true'></ejs-pivotview></div>`
})
export class AppComponent implements OnInit {
    public width?: string;
    public height?: number;
    public dataSourceSettings?: IDataOptions;
    @ViewChild('pivotview', { static: false })
    public pivotGridObj?: PivotView;
    aggregateCellInfo(args: any) {
        if (args.fieldName === 'Sold') {
            args.value = this.secondsToHms(args.value);
        }
    }
    secondsToHms(d: number) {
        d = Number(d);
        var h = Math.floor(d / 3600);
        var m = Math.floor((d % 3600) / 60);
        var s = Math.floor((d % 3600) % 60);
        return (
            ('0' + h).slice(-2) + ':' + ('0' + m).slice(-2) + ':' + ('0' +
s).slice(-2)
        );
    }
    ngOnInit(): void {
        this.width = "100%";
        this.height = 350;
        this.dataSourceSettings = {
            expandAll: false,
            dataSource: Pivot_Data as IDataset[],
            columns: [{ name: 'Year' }, { name: 'Quarter' }],
            values: [{ name: 'Sold' }, { name: 'Amount' }],
            rows: [{ name: 'Country' }, { name: 'Products' }]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Predefined Dialogs

Getting started with Angular Predefined dialogs component

This section explains how to create a simple predefined dialogs and demonstrate the basic usage of the predefined dialogs in an angular environment.

Dependencies

```
`javascript
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-buttons
\`
```

Setup an Angular environment

You can use the [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

```
`javascript
npm install -g @angular/cli
\`
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
`javascript
ng new my-app
cd my-app
\`
```

Installing Syncfusion Popups package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(>=20.2.36) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-popups](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-popups --save
`
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-popups@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-popups@ngcc --save
`
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-popups:"20.2.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Dialog module

Once you have successfully installed the popups package, the component modules are ready to configure in your application from the installed location. Syncfusion Angular package provides two different types of ngModules.

Refer to [Ng-Module](#) to learn about ngModules.

Refer to the following snippet to import the `DialogModule` in `app.module.ts` from the `@syncfusion/ej2-angular-popups`.

```
`javascript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
// Imported syncfusion DialogModule from popups package
import { DialogModule } from '@syncfusion/ej2-angular-popups';
@NgModule({
  declarations: [
    AppComponent
  ],
```

```

imports: [
  BrowserModule,
  AppRoutingModule,
  // Registering EJ2 Dialog Module
  DialogModule
],
providers: [],
bootstrap: [AppComponent]
})
export class AppModule { }
`

```

Adding CSS reference

Add dialog component's styles as given below in styles.css

```

`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-icons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-popups/styles/material.css';
`

```

Render a dialog using utility functions

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 header property.
content	Specifies the value that can be displayed in dialog's content area like the content property.
isModal	Specifies the Boolean value whether the dialog can be displayed as modal or non-modal. For more details, refer to the isModal property.
position	Specifies the value where the alert or confirm dialog is positioned within the document. For more details, refer to the position 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 used to display an errors, warnings, and information alerts that needs user awareness. The alert dialog is displayed along with the OK button. When user clicks on 'OK' button, alert dialog will get closed. Use the following code to render a simple alert dialog in an application.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button><span id="statusText"></span>`,
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public dialogObj : any;
  public alertBtnClick = (): void => {
    document.getElementById('statusText')!.style.display = 'none';
    this.dialogObj = DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width : '250px',
      okButton: { click: this.alertOkAction.bind(this) },
    });
  };
};
```

```
private alertOkAction(): void {
    this.dialogObj.hide();
    document.getElementById('statusText')!.innerHTML =
        'The user closed the Alert dialog.';
    document.getElementById('statusText')!.style.display = 'block';
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Confirm dialog

A confirm dialog box used to displays a specified message along with the 'OK' and 'Cancel' button. It is used to get approval from the user, and it appears before any critical action. After get approval from the user the dialog will disappear automatically. Use the following code to render a simple confirm dialog in an application.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
    imports: [
        DialogModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<button ej-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button><span id="statusText"></span>`
})
export class AppComponent implements OnInit {
    ngOnInit(): void {
        throw new Error('Method not implemented.');
```

```
    }
    public dialogObj :any;
    public confirmBtnClick = (): void => {
        document.getElementById('statusText')!.style.display = 'none';
        this.dialogObj = DialogUtility.confirm({
            title: 'Delete Multiple Items',
            content: 'Are you sure you want to permanently delete these items?',
            width: '300px',
            okButton: { click: this.confirmOkAction.bind(this) },
            cancelButton: { click: this.confirmCancelAction.bind(this) }
        });
    };
};
```

```

private confirmOkAction(): void {
    this.dialogObj.hide();
    document.getElementById('statusText')!.innerHTML =
        'The user confirmed the dialog box';
    document.getElementById('statusText')!.style.display = 'block';
}
private confirmCancelAction(): void {
    this.dialogObj.hide();
    document.getElementById('statusText')!.innerHTML =
        'The user canceled the dialog box.';
    document.getElementById('statusText')!.style.display = 'block';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Prompt dialog

A prompt dialog is used to get the input from the user. When the user clicks the 'OK' button the input value from the dialog is returned. If the user clicks the 'Cancel' button the null value is returned. After getting the input from the user the dialog will disappear automatically.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button><span id="statusText"></span>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

        content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
        okButton: { click: this.promptOkAction.bind(this) },
        cancelButton: { click: this.promptCancelAction.bind(this) },
    });
};
private promptOkAction(): void {
    let value: string;
    value = (document.getElementById('inputEle') as any).value;
    if (value == '') {
        this.dialogObj.hide();
        document.getElementById('statusText')!.innerHTML =
            'The user\'s input is returned as' + ' ' + ' ';
        document.getElementById('statusText')!.style.display = 'block';
    } else {
        this.dialogObj.hide();
        document.getElementById('statusText')!.innerHTML =
            'The user\'s input is returned as' + ' ' + value;
        document.getElementById('statusText')!.style.display = 'block';
    }
}
private promptCancelAction(): void {
    this.dialogObj.hide();
    document.getElementById('statusText')!.innerHTML =
        'The user canceled the prompt dialog';
    document.getElementById('statusText')!.style.display = 'block';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Draggable in Angular Predefined dialogs component

The predefined dialogs supports dragging within its target container by grabbing the dialog header, which allows the user to reposition the dialog dynamically by using `isDraggable` property.

Alert dragging**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
      DialogModule,
      ButtonModule
  ],

```



```

standalone: true,
  selector: 'app-root',
  template: `<button ejs-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width : '250px',
      isDraggable : true
    });
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Confirm drag

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public confirmBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',

```

```

        isDraggable : true,
    });
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Prompt drag

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public promptBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      isDraggable : true,
      width: '300px'
    });
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Animation in Angular Predefined dialogs component

The predefined dialogs can be animated during the open and close actions. Also, user can customize animation's **delay**, **duration** and **effect** of animation by using the **animationSettings** property.

In the below sample, **Zoom** effect is enabled. So, The Dialog will open with **ZoomIn** and close with **ZoomOut** effects.

Alert animation

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
    }
    public alertBtnClick = (): void => {
      DialogUtility.alert({
        title: 'Low Battery',
        content: '10% of battery remaining',
        width : '250px',
        animationSettings: { effect: 'Zoom' }
      });
    };
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Confirm animation

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
```

```
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public confirmBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      animationSettings: { effect: 'Zoom' }
    });
  };
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Prompt animation

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
```

```

ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

}
public promptBtnClick = (): void => {
    DialogUtility.confirm({
        title: 'Join Chat Group',
        width: '300px',
        content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
        animationSettings: { effect: 'Zoom' }
    });
};
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Position in Angular Predefined dialogs component

Customize the dialog position by using the **position** property. The position can be represented with specific **X** and **Y** values.

- The **PositionDataModel.X** can be configured with a left, center, right, or offset value. By default, the value is set as **center**.
- The **PositionDataModel.Y** can be configured with a top, center, bottom, or offset value. By default, the value is set as **center**.

Alert position

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
    imports: [
        DialogModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<button ejs-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
    ngOnInit(): void {
        throw new Error('Method not implemented.');
```

```

    }
    public alertBtnClick = (): void => {
    DialogUtility.alert({
        title: 'Low Battery',
        content: '10% of battery remaining',
        width : '250px',
        position: { X: 'center', Y: 'center' }
    });
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

[Confirm position](#)**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public confirmBtnClick = (): void => {
  DialogUtility.confirm({
    title: 'Delete Multiple Items',
    content: 'Are you sure you want to permanently delete these items?',
    width: '300px',
    position: { X: 'center', Y: 'center' }
  });
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';

```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Prompt position

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public promptBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      width: '300px',
      position: { X: 'center', Y: 'center' }
    });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Dimension in Angular Predefined dialogs component

Customize the predefined dialogs dimensions using the **height** and **width** properties.

You can specify the dimension values in both pixels and percentage format to change the default dialog width and height values.

Alert dimension

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'Not enough space',
      content: 'Delete certain files to free up space to store more items.',
      width : '250px',
      height: '200px'
    });
  };
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Confirm dimension

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',

```



```

    template: `<button ej-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
  })
  export class AppComponent implements OnInit {
    ngOnInit(): void {
      throw new Error('Method not implemented.');
```

```

    }
    public confirmBtnClick = (): void => {
      DialogUtility.confirm({
        title: 'Delete Multiple Items',
        content: 'Are you sure you want to permanently delete these items?',
        width: '300px',
        height: '200px'
      });
    }
  };
}
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Prompt dimension

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [

    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
  })
  export class AppComponent implements OnInit {
    ngOnInit(): void {
      throw new Error('Method not implemented.');
```

```

    }
    public promptBtnClick = (): void => {
      DialogUtility.confirm({
        title: 'Join Chat Group',
        content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
        height: '250px',
        width: '300px',
```

```

    });
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Max-width and max-height

To have a restricted max-width and max-height dialog dimension, you need to specify the max-width, max-height CSS properties for the component's container element by using the `cssClass` property. The max-height value is calculated in source level and set to the dialog. so, need to override the max-height property.

Use the following code to customize the max-width and max-height for alert dialog:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Series',
      content: 'In the Succinctly series, Syncfusion created a robust, free library of more than 130 technical e-books formatted for PDF, Kindle, and EPUB. Each title in the Succinctly series is written by a carefully chosen expert and provides essential content in about 100 pages. The Succinctly series was born in 2012 out of a desire to provide concise technical e-books for software developers.',
      cssClass: 'customClass'
    });
  };
};
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Min-width and min-height

To have a restricted min-width and min-height dialog dimension, you need to specify the min-width, min-height CSS properties for the component's container element by using the `cssClass` property.

Use the following code to customize the min-width and min-height for alert dialog:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'About SYNCFUSION Succinctly Series',
      content: 'The Succinctly series was born in 2012.',
      cssClass : 'customClass'
    });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customization in Angular Predefined dialogs component

You can customize the predefined dialogs buttons by using below properties.

- **okButton** - Use this property to customize **OK** button text.
- **cancelButton** - Use this property to customize **Cancel** button text.

Use the following code snippet for **alert**, **confirm** and **prompt** to customize the predefined dialogs action buttons.

For alert dialog , customized the default dialog button content as **Dismiss** by using the **text** property.

For confirm dialog, customized the default dialog buttons content as **Yes** and **No** by using the **text** property and also customized the dialog button icons by using **icon** property.

For prompt dialog , customized the default dialog buttons content as **Connect** and **Close** by using **text** property.

Alert action button

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width : '250px',
      okButton: { text: 'Dismiss' }
    });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Confirm action buttons

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public confirmBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Delete Multiple Items',
      content: 'Are you sure you want to permanently delete these items?',
      width: '300px',
      okButton: { text: 'Yes', icon: 'e-icons e-check' },
      cancelButton: { text: 'No', icon: 'e-icons e-close' }
    });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Prompt action buttons

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
```

```
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public promptBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
      width: '300px',
      okButton: { text: 'Connect' },
      cancelButton: { text: 'Close' }
    });
  };
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Show or hide dialog close button

When rendering the predefined dialogs through utility methods, You can close the dialog using the following ways. The default values of `closeOnEscape` and `showCloseIcon` is `false`.

- 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.

Use the following code for **alert**, **confirm** and **prompt** to demonstrates the different ways of hiding the utility dialog.

Alert dialog close button

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
```

```
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-danger" #alertButton
(click)="alertBtnClick()">Alert</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public alertBtnClick = (): void => {
    DialogUtility.alert({
      title: 'Low Battery',
      content: '10% of battery remaining',
      width : '250px',
      showCloseIcon : true,
      closeOnEscape : true
    });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Confirm dialog close button

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-success" #confirmButton
(click)="confirmBtnClick()">Confirm</button>`
})
export class AppComponent implements OnInit {
```

```

ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```

}
public confirmBtnClick = (): void => {
    DialogUtility.confirm({
        title: 'Delete Multiple Items',
        content: 'Are you sure you want to permanently delete these items?',
        width: '300px',
        showCloseIcon : true,
        closeOnEscape : true
    });
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Prompt dialog close button

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
    imports: [

        DialogModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<button ejs-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
    ngOnInit(): void {
        throw new Error('Method not implemented.');
```

```

    }
    public promptBtnClick = (): void => {
        DialogUtility.confirm({
            title: 'Join Chat Group',
            content:
                '<p>Enter your name:</p> <input id= "inputEle" type="text"
name="Required" class="e-input" placeholder="Type here.." />',
            width: '300px',
            showCloseIcon : true,
            closeOnEscape : true
        });
    };
}

```



```
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize dialog content

You can load custom content in predefined dialogs using the `content` property.

Use the following code to customize the dialog content to render the custom TextBox component inside the prompt dialog to get the username from the user.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DialogUtility } from '@syncfusion/ej2-popups';
@Component({
  imports: [
    DialogModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button [isPrimary]="true" #promptButton
(click)="promptBtnClick()">Prompt</button>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    throw new Error('Method not implemented.');
```

```
  }
  public promptBtnClick = (): void => {
    DialogUtility.confirm({
      title: 'Join Chat Group',
      content: '<p>Enter your name:</p><input class="e-input"
placeholder="Type here.." />',
      width: '300px'
    });
  };
};
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ProgressBar

Getting started with Angular Progress bar component

This section explains you the steps required to create a progressbar and demonstrate the basic usage of the progressbar control.

Dependencies

Below is the list of minimum dependencies required to use the progressbar component.

```
`javascript
|-- @syncfusion/ej2-angular-progressbar
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data: "*"
|-- @syncfusion/ej2-svg-base
\`
```

Installation and Configuration

- You can use **Angular CLI** to setup your angular applications.

```
`shell
npm install -g @angular/cli
\`
```

For more information, refer to [Angular sample setup](#)

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
ng new my-app
cd my-app
\`
```

Installing Syncfusion progressbar package

- Install progressbar packages using below command.

```
`javascript
npm install @syncfusion/ej2-angular-progressbar --save
\`
```

The above package installs **progressbar dependencies** which are required to render the component in Angular environment

Registering progressbar Module

- Import Chart module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-progressbar`

[src/app/app.module.ts].

```
`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the ProgressBarModule for the Chart component
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-ng-progressbar module into NgModule
  imports: [ BrowserModule, ProgressBarModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

- Modify the template in `app.component.ts` file to render the `ej2-ng-progressbar` component

[src/app/app.component.ts].

```
`javascript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the Charts component
  template: <ejs-progressbar id='percentage'></ejs-progressbar>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
```

Now use the **app-container** in the index.html instead of default one.

```
`html
```

```
<app-container></app-container>
```

```
,
```

- Now run the application in the browser using the below command.

```
,
```

```
npm start
```

```
,
```

The below example shows a basic Progressbar.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  // specifies the template string for the Progressbar component
  template: `<ejs-progressbar id='percentage' type='Linear' height='160'
[value]='value' [animation]='animation'> </ejs-progressbar>`
})
export class AppComponent {
  public animation: AnimationModel = { enable: true, duration: 2000, delay:
0 };
  public value: number = 40;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Types in Angular Progress bar component

Visualize progress in different shapes (rectangle, circle, and semi-circle) to give a unique appearance to your app design.

Linear

Set **type** to Linear to get the linear progress bar. It also support secondary progress and different mode of progress.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='determinate' type='Linear' height='60' value=100
[animation]='animation'>
    </ejs-progressbar>
    <ejs-progressbar id='indeterminate' type='Linear' height='60'
value=20
    [isIndeterminate]='isIndeterminate' [animation]=' animation'>
    </ejs-progressbar>
    <ejs-progressbar id='buffer' type='Linear' height='60' value=40
secondaryProgress=60
    [animation]=' animation'>
    </ejs-progressbar>
    <ejs-progressbar id='segment' type='Linear' height='60' value=100
segmentCount=8
    [animation]=' animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public isIndeterminate?: boolean;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.isIndeterminate = true;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Circular

Set **type** to Circular to get the circular progress bar. It also support secondary progress and different mode of progress.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';

```

```

@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='determinate' type='Circular' height='60'
value=100 [animation]='animation'>
    </ejs-progressbar>
    <ejs-progressbar id='indeterminate' type='Circular' height='60'
value=20
    [isIndeterminate]='isIndeterminate' [animation]=' animation'>
    </ejs-progressbar>
    <ejs-progressbar id='buffer' type='Circular' height='60' value=40
secondaryProgress=60
    [animation]=' animation'>
    </ejs-progressbar>
    <ejs-progressbar id='segment' type='Circular' height='60' value=100
segmentCount=8
    [animation]=' animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public isIndeterminate?: boolean;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.isIndeterminate = true;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip in Angular Progress bar component

Tooltip

The tooltip for the progress bar is used to represent the progress value. During the initial load, it can be enabled by using the [enable](#) property. The [showTooltipOnHover](#) property can show the tooltip on mouseover.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule

```

```

    ],
    standalone: true,
    selector: 'my-app',
    template:
      `<ejs-progressbar id='percentage' type='Linear' height='90' value=90
[animation]='animation' [tooltip]='tooltip'>
      </ejs-progressbar>`
  ))
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public tooltip: Object = {
    enable: true
  };
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Format

By default, the tooltip shows information about progress. In addition to that, show more information in the tooltip using the [format](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Linear' height='90' value=90
[animation]='animation' [tooltip]='tooltip'>
    </ejs-progressbar>`
  ))
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public tooltip: Object = {
    enable: true,
    format: "Progress: ${value}"
  };
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}

```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

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.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Linear' height='90' value=90
    [animation]='animation' [tooltip]='tooltip'>
      </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public tooltip: Object = {
    enable: true,
    format: "Progress: ${value}",
    textStyle: {
      fontWeight: '900',
      size: '15px',
      color: 'red',
      fontFamily: 'Roboto',
      fontStyle: 'Italic'
    }
  };
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```



```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

States in Angular Progress bar component

Visualize progress in different modes.

Determinate

This is the default state. You can use it when the progress estimation is known.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Linear' height='60' value=100>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Indeterminate

By enabling the **IsIndeterminate** property, the state of the progress bar can be changed to indeterminate when the progress cannot be estimated or is not being calculated. It can be combined with determinate mode to know that the application is estimating progress before the actual progress starts.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
```

```

    template:
      `<ejs-progressbar id='percentage' type='Linear' height='60' value=20
[isIndeterminate]='isIndeterminate' [animation]='animation'>
      </ejs-progressbar>`
  })
  export class AppComponent implements OnInit {
    public isIndeterminate?: boolean;
    public animation?: AnimationModel;
    ngOnInit(): void {
      this.animation = {enable: true};
      this.animation = { enable: true, duration: 2000, delay: 0 };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Buffer

You can use a secondary progress indicator when the primary progress depends on the secondary progress. This will allow users to visualize both primary and secondary progress simultaneously.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Linear' height='60' value=40
secondaryProgress=60 [animation]='animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customization in Angular Progress bar component

Segments

We can divide a progress bar into multiple segments using a `segmentCount` to visualize the progress of multiple sequential tasks.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Circular' height='60'
    segmentCount=8 value=100 [animation]='animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Thickness

Customize the thickness of the track using [trackThickness](#), progress using [progressThickness](#) and secondary progress using [secondaryProgressThickness](#) to render the progress bar with different appearances.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel, FontModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
```

```

    ],
    standalone: true,
    selector: 'my-app',
    template:
      `<ejs-progressbar id='percentage' type='Linear' height='60' width='90%'
trackThickness=24 progressThickness=24 secondaryProgressThickness=20
value=100 [showProgressValue]='showProgressValue' [labelStyle]='labelStyle'
[animation]='animation'>
      </ejs-progressbar>`
  ))
export class AppComponent implements OnInit {
  public showProgressValue?: boolean;
  public labelStyle?: FontModel;
  public animation?: AnimationModel;
  ngOnInit(): void {
    this.labelStyle = { color: '#FFFFFF' };
    this.showProgressValue = true;
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Min, Max, and Value

The progress bar value is set by using the **value** property in progress bar. The minimum and maximum value of the progress bar can be set by using the **min** and **max** property in the progress bar respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel, FontModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar #linear id='linear' [type]='type' [maximum]='max'
[minimum]='min' [value]='value' [width]='width' [height]='height'></ejs-
progressbar>`
  ))
export class AppComponent implements OnInit {
  public type?: string;
  public width?: string;
  public height?: string;
  public min?: number;

```

```

public max?: number;
public value?: number;
public animation?: AnimationModel;
ngOnInit(): void {
  this.type = "Linear";
  this.width = "100%";
  this.height = "60";
  this.min = 0;
  this.max = 100;
  this.value = 70;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Radius

The radius of the progress bar can be customized using **radius** property and corner can be customized by **cornerRadius** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar';
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Circular' height='160px'
width='160px' trackColor='#FFD939' radius='100%'
progressColor='white' cornerRadius='Round' trackThickness=80
progressThickness=10 value=60 [animation]='animation'>
</ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

InnerRadius

The inner radius of the progress bar can be customized using `innerRadius` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-progressbar id='percentage' type='Circular'
height='160px' width='160px' trackColor='#FFD939'
secondaryProgressColor='green' radius='100%' innerRadius='70%'
progressColor='white' cornerRadius='Round' trackThickness=80
progressThickness=10 value=60 [animation]='animation'></ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Progress color and track color

Customize the color of progress, secondary progress, and track by using the [progressColor](#), [secondaryProgressColor](#), and [trackColor](#) properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { FontModel, AnimationModel, ITextRenderEventArgs } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
```

```

standalone: true,
  selector: 'my-app',
  template: `<ejs-progressbar id='percentage' type='Linear' height='60'
width='90%' trackThickness=24 progressThickness=24 secondaryProgress=60
value = 50 secondaryProgressColor='green' progressColor='#E3165B'
trackColor='#F8C7D8' [labelStyle]='labelStyle'
(textRender)='textRender($event)' [showProgressValue]='showProgressValue'
[animation]='animation'>
    </ejs-progressbar>`
  })
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public labelStyle?: FontModel;
  public showProgressValue?: boolean;
  public textRender(args: ITextRenderEventArgs): void {
    args.text = '50';
  }
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.labelStyle = { color: '#FFFFFF' };
    this.showProgressValue = true;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Annotation in Angular Progress bar component

Annotation

In the circular progress bar, you can add any view to the center using the **Content** property in annotation.

For example, you can include add, start, or pause button to control the progress. You can also add an image that indicates the actual task in progress or add custom text that conveys how far the task is completed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule, ProgressAnnotationService } from
'@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ProgressBarModule
  ],
  providers: [ProgressAnnotationService],
  standalone: true,
  selector: 'my-app',
  template:

```

```

    <ejs-progressbar id='percentage' type='Circular' trackColor='#FFD939'
    cornerRadius='Round'
        innerRadius='190%' [trackThickness]='trackThickness'
    >
    <e-progressbar-annotations>
        <e-progressbar-annotation
            content='<div id="point1" style="font-size:20px;font-
weight:bold;color:#ffffff;fill:#ffffff"><span>60%</span></div>'>
        </e-progressbar-annotation>
    </e-progressbar-annotations>
    </ejs-progressbar>`
    })
    export class AppComponent implements OnInit {
        public trackThickness?: number;
        ngOnInit(): void {
            this.trackThickness = 80;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label

You can show the progress value in both linear and circular progress bar using **showProgressValue** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ProgressBarModule, ProgressAnnotationService } from
'@syncfusion/ej2-angular-progressbar';
import { Component, OnInit } from '@angular/core';
import { FontModel, AnimationModel, ITextRenderEventArgs } from
'@syncfusion/ej2-progressbar';
@Component({
    imports: [
        ProgressBarModule
    ],
    providers: [ProgressAnnotationService],
    standalone: true,
    selector: 'my-app',
    template:
`<ejs-progressbar id='percentage' type='Linear'
[trackThickness]='trackThickness' [progressThickness]='progressThickness'
[value]='value' [labelStyle]='labelStyle'
(textRender)='textRender2($event)' [showProgressValue]='showProgressValue'
[animation]='animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {

```



```

    public animation?: AnimationModel;
    public value?: number;
    public trackThickness?: number;
    public progressThickness?: number;
    public labelStyle?: FontModel;
    public showProgressValue?: boolean;
    public textRender2(args: ITextRenderEventArgs): void {
        args.text = '50';
    }
    ngOnInit(): void {
        this.trackThickness = 24;
        this.progressThickness = 24;
        this.value = 50;
        this.animation = { enable: true, duration: 2000, delay: 0 };
        this.labelStyle = { color: '#FFFFFF' };
        this.showProgressValue = true;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Animation in Angular Progress bar component

<!-- markdownlint-disable MD033 -->

Progress Bar support to animate the progress by using **animation** property. Enable the animation by setting **enable** property and also you can control the speed by using **duration** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ProgressBarModule, ProgressAnnotationService } from '@syncfusion/ej2-angular-progressbar';
import { Component, OnInit } from '@angular/core';
import { AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
    imports: [
        ProgressBarModule
    ],
    providers: [ProgressAnnotationService],
    standalone: true,
    selector: 'my-app',
    template: `
        <ejs-progressbar id='percentage' type='Linear' value=100
        [animation]='animation'>
        </ejs-progressbar>
    `
})
export class AppComponent implements OnInit {
    public animation?: AnimationModel;
}

```

```
ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Range in Angular Progress bar component

<!-- markdownlint-disable MD033 -->

Range represents the entire span of the ProgressBar and can be defined using the **minimum** and **maximum** properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { FontModel, AnimationModel } from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <ejs-progressbar id='percentage' type='Linear' trackThickness=24
    progressThickness=24 value = 90 minimum=10 maximum=90
    [labelStyle]='labelStyle' [showProgressValue]='showProgressValue'
    [animation]='animation'>
    </ejs-progressbar>`
  })
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public labelStyle?: FontModel;
  public showProgressValue?: boolean;
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.labelStyle = { color: '#FFFFFF' };
    this.showProgressValue = true;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Events in Angular Progress bar component

Value Change

<!-- markdownlint-disable MD033 -->

valueChanged event is triggered when the progress value is changed.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ProgressBar, FontModel, AnimationModel, IProgressValueEventArgs }
from '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar #charts id='charts' type='Linear' trackThickness=24
progressThickness=24 value = 90 [labelStyle]='labelStyle'
(valueChanged)='valueChanged($event)'
[showProgressValue]='showProgressValue' [animation]='animation'>
    </ejs-progressbar>
    <button id="reLoad" (click)="onClick()">ValueChanged</button>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public labelStyle?: FontModel;
  public showProgressValue?: boolean;
  @ViewChild('charts')
  public charts?: ProgressBar;
  public valueChanged(args: IProgressValueEventArgs): void {
    args.progressColor = '#2BB20E';
  }
  public onClick = () => {
    this.charts!.value = 50;
  }
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.labelStyle = { color: '#FFFFFF' };
    this.showProgressValue = true;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ProgressCompleted

<!-- markdownlint-disable MD033 -->

ProgressCompleted event is triggered when the progress attains the Maximum value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressBarModule } from '@syncfusion/ej2-angular-progressbar'
import { Component, OnInit } from '@angular/core';
import { FontModel, AnimationModel, IProgressValueEventArgs } from
 '@syncfusion/ej2-progressbar';
@Component({
  imports: [
    ProgressBarModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-progressbar id='percentage' type='Linear' trackThickness=24
progressThickness=24 value = 100 [labelStyle]='labelStyle'
(progressCompleted)='progressCompleted($event)'
[showProgressValue]='showProgressValue' [animation]='animation'>
    </ejs-progressbar>`
})
export class AppComponent implements OnInit {
  public animation?: AnimationModel;
  public labelStyle?: FontModel;
  public showProgressValue?: boolean;
  public progressCompleted(args: IProgressValueEventArgs): void {
    args.progressColor = '#2BB20E';
  }
  ngOnInit(): void {
    this.animation = { enable: true, duration: 2000, delay: 0 };
    this.labelStyle = { color: '#FFFFFF' };
    this.showProgressValue = true;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Accessibility in Angular Progress bar component

The Progress bar 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 Progress bar 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 Progress bar component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Progress bar component:

- progressbar (role)
- aria-valuemin (attribute)
- aria-valuemax (attribute)
- aria-valuenow (attribute)
- aria-label (attribute)

Keyboard interaction

The Progress bar 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 Progress bar component.

| **Press** | **To do this** |

| --- | --- |

| **Tab** | Moves the focus to the Progress bar element. |

| **Ctrl + P** | Prints the Progress bar. |

Ensuring accessibility

The Progress bar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Progress bar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Progress bar component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

ProgresButton

Getting started with Angular Progress button component

This section explains how to create a simple ProgressButton and demonstrate the basic usage of the ProgressButton component in an Angular environment.

Dependencies

The list of dependencies required to use the ProgressButton component in your application is given as follows:

```
`typescript
|-- @syncfusion/ej2-angular-splitbuttons
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
\`
```

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

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

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
ng new my-app  
cd my-app
```

Installing Syncfusion ProgressButton package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(`>=20.2.36`) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-splitbuttons](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-splitbuttons --save
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-splitbuttons@ngcc](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-splitbuttons@ngcc --save
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash  
@syncfusion/ej2-angular-splitbuttons:"20.2.38-ngcc"
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding ProgressButton module

Import ProgressButton module into Angular application(app.module.ts) from the package

`@syncfusion/ej2-angular-splitbuttons`.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Imported Syncfusion Progress button module from split buttons package
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, ProgressButtonModule ], // Registering EJ2 ProgressButtonModule.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding Syncfusion ProgressButton component

Modify the template in `app.component.ts` file to render the ProgressButton component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To render progress button. -->
<button ej2-progressbutton content='Spin Left'></button>`
})
export class AppComponent {
}
```

Adding CSS reference

Add ProgressButton component's styles as given below in `style.css`.

```
`css
@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';
```

```
,
```

Running the application

Run the application in the browser using the following command:

```
,
```

```
ng serve
```

```
,
```

The below example shows a basic ProgressButton component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    ProgressButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render progress button. -->
    <button ej2-progressbutton content='Spin
Left'></button></div>`
})
export class AppComponent {
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

ProgressButton supports different styles, types and sizes like [Button](#). In addition, it also supports **top** and **bottom** icon positions.

Enable progress in button

You can enable the background filler UI by setting the [enableProgress](#) property to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
@Component({
  imports: [
```

```

        ProgressButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render progress button. -->
        <button ej-progressbutton content='Spin Left'
[enableProgress]='true'></button></div>`
    })
    export class AppComponent {
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD002 MD022 -->

Spinner and progress in Angular Progress Button component

Change spinner position

Spinner position can be changed by modifying the [position](#) property of [spinSettingsModel](#). By default, the spinner is positioned at the left of the ProgressButton. You can position it at the [left](#), [right](#), [top](#), [bottom](#), or [center](#) of the text content.

Change spinner size

Spinner size can be changed by modifying the [width](#) property of [spinSettingsModel](#). In this demo, the [width](#) is set to [20](#) to change the spinner size.

Spinner template

You can use custom spinner by specifying the [template](#) property of [spinSettingsModel](#) with custom styles.

The following sample demonstrates the above functionalities of the spinner.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { SpinSettingsModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

        ProgressButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render progress button. -->

```

```

        <button ejs-progressbutton content='Submit'
[spinSettings]='spinSettings'></button></div>`
    })
    export class AppComponent {
        public spinSettings : SpinSettingsModel = { position: 'Right', width:
20, template: '<div class="template"></div>' };
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Progress*Content animation*

The [content](#) of the ProgressButton can be animated during progress using the [effect](#) property of [animationSettingsModel](#). You can also set custom duration and timing function using the [duration](#) and [easing](#) properties. The possible [effect](#) values are [None](#), [SlideLeft](#), [SlideRight](#), [SlideUp](#), [SlideDown](#), [ZoomIn](#), and [ZoomOut](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { SpinSettingsModel, AnimationSettingsModel } from '@syncfusion/ej2-
angular-splitbuttons';
@Component({
    imports: [

        ProgressButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <button ejs-progressbutton content='Slide Left'
[enableProgress]='true' [animationSettings]= 'animationSettings'
[spinSettings]='spinSettings'></button></div>`
    })
    export class AppComponent {
        public spinSettings : SpinSettingsModel = { position: 'Center' };
        public animationSettings : AnimationSettingsModel = {
effect: 'SlideLeft', duration: 500, easing: 'linear' };
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Change step of the ProgressButton

The progress can be visualized at the specified interval by changing the [step](#) property in the [begin](#) event of the ProgressButton. In this demo, the [step](#) property is set to **20** to show progress at every 20% increment.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { ProgressEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    ProgressButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <button ejs-progressbutton content='Progress Step'
[enableProgress]='true' (begin)='begin($event)' cssClass='e-hide-
spinner'></button></div>`
})
export class AppComponent {
  public begin(args: ProgressEventArgs): void {
    args.step = 20;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The class `e-hide-spinner` hides the spinner in the ProgressButton, For more information, see [hide spinner](#) section.

Change progress dynamically

The progress can be changed dynamically by modifying the [percent](#) property in the ProgressButton events. In this demo, on 40% completion of progress, the [percent](#) property is set to **90** to show dynamic change of the progress.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
import { ProgressEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
```

```

imports: [
    ProgressButtonModule
],
standalone: true,
selector: 'app-root',
template: `<div class="e-section-control">
    <button ejs-progressbutton [content]='content'
[enableProgress]='true' [duration]='15000' (begin)='begin($event)'
(progress)='progress($event)' (end)='end($event)' cssClass='e-hide-
spinner'></button></div>`
})
export class AppComponent {
    public content: string = 'Progress';
    public begin(args: ProgressEventArgs): void {
        this.content = 'Progress ' + args.percent + '%';
    }
    public progress(args: ProgressEventArgs): void {
        this.content = 'Progress ' + args.percent + '%';
        if (args.percent === 40) {
            args.percent = 90;
        }
    }
    public end(args: ProgressEventArgs): void {
        this.content = 'Progress ' + args.percent + '%';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The method [dataBind](#) applies the property changes immediately to the component.

Start and stop methods

You can pause and resume the progress using the [stop](#) and [start](#) methods, respectively. In this demo, clicking the ProgressButton will pause and resume the progress.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component, ViewChild } from '@angular/core';
import { ProgressButtonComponent } from '@syncfusion/ej2-angular-
splitbuttons';
@Component({
imports: [
    ProgressButtonModule
],
standalone: true,
selector: 'app-root',

```

```

    template: `<div class="e-section-control">
        <button #progressBtn ejs-progressbutton [content]='content'
[enableProgress]='true' [duration]='4000' (end)='end()' [iconCss]='iconCss'
cssClass='e-hide-spinner' (click)="btnClick()"></button></div>`
    })
    export class AppComponent {
        @ViewChild('progressBtn')
        public progressBtn : ProgressButtonComponent | any;
        public content: string = 'Download';
        public iconCss: string = 'e-btn-sb-icon e-download';
        public end(): void {
            this.content = 'Download';
            this.iconCss = 'e-btn-sb-icon e-download';
        }
        public btnClick(): void {
            if(this.content === 'Download') {
                this.content = 'Pause';
                this.iconCss = 'e-btn-sb-icon e-pause';
            }
            else if(this.content === 'Pause') {
                this.content = 'Resume';
                this.iconCss = 'e-btn-sb-icon e-play';
                this.progressBtn.stop();
            }
            else if(this.content === 'Resume') {
                this.content = 'Pause';
                this.iconCss = 'e-btn-sb-icon e-pause';
                this.progressBtn.start();
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [How to hide spinner](#)
- [Customize ProgressButton using cssClass](#)

Accessibility in Angular Progress button component

The Progress 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 Progress 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 Progress button component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Progress button component:

| Attributes | Purpose |

| --- | --- |

| `aria-label` | Provides an accessible name for the icon only Progress button. |

| `aria-disabled` | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

Keyboard interaction

The Progress 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 Progress button component.

| **Press** | **To do this** |

| --- | --- |

| **Enter / Space** | **Starts the progress.** |

Ensuring accessibility

The Progress button component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Progress button component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Progress button component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Change the text content and styles of the progressbutton during progress in Angular Progress button component

You can change the text content and styles of the ProgressButton during progress by changing the text content and the [cssClass](#) property at the [begin](#) and [end](#) events.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    ProgressButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render progress button. -->
    <button ej2-progressbutton [content]='content'
[cssClass]='cssClass' [enableProgress]='true' [duration]='4000'
(begin)='begin()' (end)='end()'></button></div>`
})
export class AppComponent {
  public content: string = 'Upload';
  public cssClass: string = 'e-hide-spinner';
  public begin(): void {
    this.content = 'Uploading...';
    this.cssClass = 'e-hide-spinner e-info';
  }
}
```



```

    public end(): void {
        this.content = 'Success';
        this.cssClass = 'e-hide-spinner e-success';
        setTimeout(() => {
            this.content = 'Upload';
            this.cssClass = 'e-hide-spinner';
        }, 500)
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize progress using cssclass in Angular Progress button component

You can customize the background filler UI using the [cssClass](#) property.

- Adding `e-vertical` to `cssClass` shows vertical progress.
- Adding `e-progress-top` to `cssClass` shows progress at the top.

You can also show reverse progress by adding custom class to the [cssClass](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    ProgressButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <button ejs-progressbutton content='Vertical Progress'
[enableProgress]='true' cssClass='e-hide-spinner e-vertical'
[duration]='4000'></button>
    <button ejs-progressbutton content='Progress Top'
[enableProgress]='true' cssClass='e-hide-spinner e-progress-top'
[duration]='4000'></button>
    <button ejs-progressbutton content='Reverse Progress'
[enableProgress]='true' cssClass='e-hide-spinner e-reverse-progress'
[duration]='4000'></button>
  </div>`
})
export class AppComponent {
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Hide spinner in Angular Progress button component

You can hide spinner in the ProgressButton by setting the `e-hide-spinner` property to [cssClass](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    ProgressButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <button ej2-progressbutton content='Progress' cssClass='e-
hide-spinner' [enableProgress]='true'></button></div>`
})
export class AppComponent {
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Trace events of progress button in Angular Progress button component

The ProgressButton component triggers events based on its actions. The events can be used as extension points to perform custom operations.

The events available in ProgressButton are [fail](#), [begin](#), [progress](#), and [end](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ProgressButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component, ViewChild } from '@angular/core';
import { ProgressEventArgs } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [

    ProgressButtonModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `
    <div class="e-section-control">
      <div class="progress-btn-section">
        <button ejs-progressbutton content='Progress'
[enableProgress]='true' (begin)='begin($event)' (end)='end($event)'
(progress)='progress($event)' (fail)='fail($event)' cssClass='e-hide-
spinner'></button>
      </div>
      <div class="property-section">
        <table id="propertyTable" title="Event trace">
          <tbody>
            <th>Event trace:-</th>
            <tr>
              <td [innerHTML]="eventTrace"></td>
            </tr>
          </tbody>
        </table>
      </div>
      <button #clear id="clear" ejs-button cssClass='e-small'
(click)='btnClick()'>Clear</button>
    </div>
  `
})
export class AppComponent {
  begin($event: any) {
    throw new Error('Method not implemented.');
```

```

  }
  public eventTrace: string = '';
  public end(args: ProgressEventArgs): void {
    this.updateEventLog(args);
  }
  public progress(args: ProgressEventArgs): void {
    this.updateEventLog(args);
  }
  public fail(args: Event): void {
    this.updateEventLog(args);
  }
  public updateEventLog(args: any): void {
    this.eventTrace = this.eventTrace + args.name + ' Event triggered.
<br />'
  }
  public btnClick(): void {
    this.eventTrace = '';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

QueryBuilder

Getting started with Angular Query builder component

This section explains how to create and demonstrate the basic usage of the [Angular Query Builder](#) module.

Dependencies

The list of dependencies required to use the Query Builder module in your application is given below:

```
`javascript
|-- @syncfusion/ej2-angular-querybuilder
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-querybuilder
|-- @syncfusion/ej2-datamanager
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-calenders
|-- @syncfusion/ej2-inputs
`,`
```

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

```
`
npm install -g @angular/cli
`,`
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
`
ng new my-app
cd my-app
`,`
```

Installing Syncfusion Query Builder package

To install Query Builder package, use the following command.

```
`
npm install @syncfusion/ej2-angular-querybuilder --save
`,`
```

The above package installs [Query Builder dependencies](#) which are required to render the component in the Angular environment.

Adding Query Builder module

Import Query Builder module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-querybuilder`.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Importing QueryBuilderModule from ej2-angular-querybuilder package.
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, QueryBuilderModule ], // Declaration of QueryBuilder module into
  NgModule.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

Adding Syncfusion Query Builder component

Modify the template in `app.component.ts` file to render the Angular Query Builder component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
<ejs-querybuilder width="70%">
<e-columns>
<e-column field="EmployeeID" label="Employee ID" type="number"></e-column>
<e-column field="FirstName" label="First Name" type="string"></e-column>
<e-column field="TitleOfCourtesy" label="Title Of Courtesy" type="boolean" [values]="values"></e-
column>
<e-column field="Title" label="Title" type="string"></e-column>
<e-column field="HireDate" label="Hire Date" type="date" format="dd/MM/yyyy"></e-column>
<e-column field="Country" label="Country" type="string"></e-column>
<e-column field="City" label="City" type="string"></e-column>
```

```

</e-columns>
</ejs-querybuilder>`
})
export class AppComponent {
  public values: string[] = ['Mr.', 'Mrs.'];
}
`

```

Adding CSS reference

Add Angular Query Builder component's styles as given below in `style.css`.

```

`css
@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-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-inputs/styles/material.css";
@import "../node_modules/@syncfusion/ej2-calendars/styles/material.css";
@import "../node_modules/@syncfusion/ej2-popups/styles/material.css";
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import "../node_modules/@syncfusion/ej2-querybuilder/styles/material.css";
`

```

Running the application

Run the application in the browser using the following command:

```

`
ng serve
`

```

The following example shows a basic Query Builder component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    QueryBuilderModule
  ],
  standalone: true,

```

```

    selector: 'app-root',
    template: `<!-- To render Query Builder. -->
                <ejs-querybuilder width="70%">
                    <e-columns>
                        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
                        <e-column field="FirstName" label="First Name"
type="string"></e-column>
                        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
                        <e-column field="Title" label="Title" type="string"></e-
column>
                        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
                        <e-column field="Country" label="Country"
type="string"></e-column>
                        <e-column field="City" label="City" type="string"></e-
column>
                    </e-columns>
                </ejs-querybuilder>`
  })
  export class AppComponent {
    public values: string[] = ['Mr.', 'Mrs.'];
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Rendering with rule

To render the Angular Query Builder component with rule, use the [rule](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
@Component({
  imports: [

    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
                <ejs-querybuilder width="70%" [rule]="importRules">
                    <e-columns>
                        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>

```

```

        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>`
    })
    export class AppComponent implements OnInit {
        public importRules?: RuleModel;
        public values: string[] = ['Mr.', 'Mrs.'];
        ngOnInit(): void {
            this.importRules = {
                'condition': 'and',
                'rules': [{
                    'label': 'Employee ID',
                    'field': 'EmployeeID',
                    'type': 'number',
                    'operator': 'equal',
                    'value': 1
                },
                {
                    'label': 'Title',
                    'field': 'Title',
                    'type': 'string',
                    'operator': 'equal',
                    'value': 'Sales Manager'
                }
            ]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can also explore our [Angular Query Builder example](#) that shows how to render the Query Builder in Angular.

Columns in Angular Query builder component

The column definitions are used as the [dataSource](#) schema in the Query Builder. This plays a vital role in rendering column values. The query builder operations such as create or delete conditions and create or delete group they are performed based on the column definitions. The [field](#) property of the columns is necessary to map the data source values in the query builder columns.

If the column field is not specified in the [dataSource](#), the column values will be empty.

Auto generation

The [columns](#) are automatically generated when the [columns](#) declaration is empty or undefined while initializing the query builder. All the columns in the [dataSource](#) are bound as the query builder columns.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { employeeData } from './datasource';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data">
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  ngOnInit(): void {
    this.data = employeeData;
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

When columns are auto-generated, the column type will be determined from the first record of the [dataSource](#).

Labels

By default, the column label is displayed from the column [field](#) value. To override the default label, you have to define the [label](#) value.

Operators

The operator for a column can be defined in the [operators](#) property. The available operators and its supported data types are:

Operators	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 Number Date Boolean |

| notequal | Checks whether the value is not equal to the specified value. | String Number Date Boolean |

| greaterthan | Checks whether the value is greater than the specified value. | Date Number |

| greaterthanorequal | Checks whether a value is greater than or equal to the specified value. | Date Number |

| lessthan | Checks whether the value is less than the specified value. | Date Number |

| lessthanorequal | Checks whether the value is less than or equal to the specified value. | Date Number |

| between | Checks whether the value is between the two-specific value. | Date Number |

| notbetween | Checks whether the value is not between the two-specific value. | Date Number |

| in | Checks whether the value is one of the specific values. | String Number |

| notin | Checks whether the value is not in the specific values. | String Number |

Step

The Query Builder allows you to set the step values to the number fields. So that you can easily access the numeric textbox. Use the [step](#) property, to set the step value for number values.

Format

The Query Builder formats date and number values. Use the [format](#) property to format date and number values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
    [dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
        type="number" step="10" [operators]="employeeOperators"></e-column>
        <e-column field="FirstName" label="First Name"
        type="string"></e-column>
```

```

        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>`
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    public employeeOperators?: Object[];
    values: any;
    ngOnInit(): void {
        this.data = employeeData;
        this.importRules = {
            'condition': 'and',
            'rules': [{
                'label': 'Employee ID',
                'field': 'EmployeeID',
                'type': 'number',
                'operator': 'equal',
                'value': 1001
            },
            {
                'label': 'Hire Date',
                'field': 'HireDate',
                'type': 'date',
                'operator': 'equal',
                'value': '07/05/1991'
            }
        ]
    };
    this.employeeOperators = [
        { value: 'equal', key: 'Equal' },
        { value: 'notequal', key: 'Not Equal' },
        { value: 'in', key: 'In' },
        { value: 'notin', key: 'Not In' }
    ];
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Validations

Validation allows you to validate the conditions and it display errors for invalid fields while using the `validateFields` method. To enable validation in the query builder , set the `allowValidation` to true. Column fields are validated after setting [allowValidation](#) as to true. So, you should manually configure the validation for Operator and, Value fields through [validation](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from './datasource';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" allowValidation="true">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number" validation="validateRule"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="validate()" >Validate Field</button>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  @ViewChild('querybuilder')
  public qryBldrObj?: QueryBuilderComponent;
  public validateRule: { [key: string]: Boolean } = { isRequired: true };
  public values: string[] = ['Mr.', 'Mrs.'];
  ngOnInit(): void {
    this.data = employeeData;
  }
}
```

```

    validate(): void {
        this.qryBldrObj!.validateFields();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set [isRequired](#) validation for **Operator** and **Value** fields.

Set [max](#), [min](#) values for number values.

Data binding in Angular Query builder component

The Query Builder 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 **DataManager** or JavaScript object array collection. It supports two kinds of binding:

- Local data
- Remote data

Local data

To bind local data to the query builder, you can assign the [dataSource](#) property with a JavaScript object array. The local data source can also be provided as an instance of the **DataManager**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from './datasource';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>

```

```

        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>`
}))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    public values: string[] = ['Mr.', 'Mrs.'];
    ngOnInit(): void {
        this.data = employeeData;
        this.importRules = {
            'condition': 'and',
            'rules': [{
                'label': 'Employee ID',
                'field': 'EmployeeID',
                'type': 'number',
                'operator': 'equal',
                'value': 1
            },
            {
                'label': 'Title',
                'field': 'Title',
                'type': 'string',
                'operator': 'equal',
                'value': 'Sales Manager'
            }
        ]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

By default, **DataManager** uses **JsonAdaptor** for local data-binding.

Remote data

To bind remote data to the query builder, assign service data as an instance of **DataManager** to the [dataSource](#) property. To interact with remote data source, provide the endpoint [url](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'

```

```

import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="Designation" label="Designation"
type="string"></e-column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: DataManager;
  importRules: { condition: string; rules: ({ label: string; field:
string; type: string; operator: string; value: number; } | { label: string;
field: string; type: string; operator: string; value: string; })[]; } |
undefined;
  values: any;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://services.syncfusion.com/js/production/api/Employees/',
      adaptor: new ODataAdaptor(),
    });
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'Employee ID',
        'field': 'EmployeeID',
        'type': 'number',
        'operator': 'equal',
        'value': 1
      },
      {
        'label': 'Designation',
        'field': 'Designation',
        'type': 'string',
        'operator': 'equal',
        'value': 'Developer'
      }
    ]
  };
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

By default, **DataManager** uses **ODataAdaptor** for remote data-binding.

Binding with 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.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: DataManager;
  public values: string[] = ['Mr.', 'Mrs.'];
```



```

importRules: { condition: string; rules: ({ label: string; field:
string; type: string; operator: string; value: number; } | { label: string;
field: string; type: string; operator: string; value: string; })[]; } |
undefined;
ngOnInit(): void {
    this.data = new DataManager({
        url: 'https://services.syncfusion.com/js/production/api/Employees/',
        adaptor: new ODataAdaptor(),
        crossDomain: true
    });
    this.importRules = {
        'condition': 'and',
        'rules': [{
            'label': 'Employee ID',
            'field': 'EmployeeID',
            'type': 'number',
            'operator': 'equal',
            'value': 1
        },
        {
            'label': 'Title',
            'field': 'Title',
            'type': 'string',
            'operator': 'equal',
            'value': 'Sales Manager'
        }
    ]
    };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Binding with 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.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
    imports: [
        QueryBuilderModule
    ],

```

```

standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: DataManager;
  public values: string[] = ['Mr.', 'Mrs.'];
  ngOnInit(): void {
    this.data = new DataManager({
      url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
      adaptor: new ODataV4Adaptor
    });
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'Employee ID',
        'field': 'EmployeeID',
        'type': 'number',
        'operator': 'equal',
        'value': 1
      },
      {
        'label': 'Title',
        'field': 'Title',
        'type': 'string',
        'operator': 'equal',
        'value': 'Sales Manager'
      }
    ]
  };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Web API

You can use **WebApiAdaptor** to bind query builder with Web API created using OData endpoint.

```
`typescript
```

```
import { Component } from '@angular/core';
```

```
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  template: `<!-- To render Query Builder. -->
```

```
    <ejs-querybuilder #querybuilder width="70%" [dataSource]="data" [rule]="importRules">
```

```
      <e-columns>
```

```
        <e-column field="EmployeeID" label="Employee ID" type="number"></e-column>
```

```
        <e-column field="FirstName" label="First Name" type="string"></e-column>
```

```
        <e-column field="TitleOfCourtesy" label="Title Of Courtesy" type="boolean" [values]="values"></e-column>
```

```
        <e-column field="Title" label="Title" type="string"></e-column>
```

```
        <e-column field="HireDate" label="Hire Date" type="date" format="dd/MM/yyyy"></e-column>
```

```
        <e-column field="Country" label="Country" type="string"></e-column>
```

```
        <e-column field="City" label="City" type="string"></e-column>
```

```
      </e-columns>
```

```
    </ejs-querybuilder>`
```

```
  })
```

```
  export class AppComponent implements OnInit {
```

```
    public data: DataManager;
```

```
    ngOnInit(): void {
```

```
      this.data = new DataManager({
```

```
        url: 'api/OrderAPI',
```

```
        adaptor: new WebApiAdaptor
```

```
      });
```

```
      this.importRules = {
```

```
        'condition': 'and',
```

```
        'rules': [{
```

```
          'label': 'Employee ID',
```

```

'field': 'EmployeeID',
'type': 'number',
'operator': 'equal',
'value': 1
},
{
'label': 'Title',
'field': 'Title',
'type': 'string',
'operator': 'equal',
'value': 'Sales Manager'
}}
};
}
}
,

```

Data Manager

You can use the created conditions in DataManager through the getPredicate method. This method creates predicates which is used as conditions in DataManager.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit, ViewChild } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from './datasource';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
    [dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
        type="number"></e-column>

```

```

        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="refreshTable()" >Get Data</button>
    <table id='datatable' class='e-table'>
        <tr><th>EmployeeID</th><th>Title</th><th>City</th></tr>
        <tr *ngFor="let item of items">
            <td>{{item.EmployeeID}}</td><td>{{item.Title}}</td><td>{{item.City}}</td>
            </tr>
        </table>`
    ))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    public values: string[] = ['Mr.', 'Mrs.'];
    public items?: Object[] | any;
    public dataManagerQuery!: Query;
    ngOnInit(): void {
        this.data = employeeData ;
        this.importRules = {
            'condition': 'and',
            'rules': [{
                'label': 'Employee ID',
                'field': 'EmployeeID',
                'type': 'number',
                'operator': 'equal',
                'value': 1
            }]
        };
    }
    refreshTable(): void {
        this.dataManagerQuery = new Query().select(['EmployeeID', 'Title',
'City']).where(this.qryBldrObj!.getPredicate(this.qryBldrObj!.rule)).take(8)
;
        this.items = [];
        this.items = new
DataManager(<JSON[]>this.data).executeLocal(this.dataManagerQuery);
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Complex Data Binding

Complex Data Binding allows you to create subfield for columns. To implement complex data binding, either bind the complex data in nested columns or specify complex data source and separator must be given in querybuilder.

In the following sample, complex data was bound in nested columns.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule, RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component, ViewChild, OnInit } from '@angular/core';
import { QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { RuleModel, RuleChangeEventArgs, ColumnsModel } from '@syncfusion/ej2-querybuilder';
import { RadioButton, ChangeEventArgs } from '@syncfusion/ej2-buttons';
import { getComponent } from '@syncfusion/ej2-base';
import { MultiSelect, CheckBoxSelection } from '@syncfusion/ej2-dropdowns';
import { RadioButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    QueryBuilderModule,
    CheckBoxModule,
    DropDownListModule,
    DropDownButtonModule,
    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: `template-driven.html`
})
export class AppComponent implements OnInit {
  @ViewChild('querybuilder') qryBldrObj: QueryBuilderComponent | undefined;
  public importRules?: RuleModel;
  public columns1?: ColumnsModel[];
  public columns2?: ColumnsModel[];
  public columns3?: ColumnsModel[];
  ngOnInit(): void {
    this.importRules = {
      condition: 'and',
      rules: [{
        label: 'ID',
        field: 'Employee.ID',
        type: 'string',
```

```

        operator: 'equal',
        value: 0
    },
    {
        label: 'Last Name',
        field: 'Name.LastName',
        type: 'string',
        operator: 'contains',
        value: 'malan'
    },
    {
        condition: 'or',
        rules: [{
            label: 'City',
            field: 'Country.State.City',
            operator: 'startswith',
            type: 'string',
            value: 'U'
        },
        {
            label: 'Region',
            field: 'Country.Region',
            operator: 'endswith',
            type: 'string',
            value: 'C'
        },
        {
            label: 'Name',
            field: 'Country.Name',
            operator: 'isnotempty'
        }
    ]
    }
];
this.columns1 = [
    { field: 'ID', label: 'ID', type: 'number' },
    { field: 'DOB', label: 'Date of birth', type: 'date' },
    { field: 'HireDate', label: 'Hire Date', type: 'date' },
    { field: 'Salary', label: 'Salary', type: 'number' },
    { field: 'Age', label: 'Age', type: 'number' },
    { field: 'Title', label: 'Title', type: 'string' }
];
this.columns2 = [
    { field: 'FirstName', label: 'First Name', type: 'string' },
    { field: 'LastName', label: 'Last Name', type: 'string' }
];
this.columns3 = [
    { field: 'State', label: 'State', columns: [
        { field: 'City', label: 'City', type: 'string' },
        { field: 'Zipcode', label: 'Zip Code', type: 'number' }
    ] },
    { field: 'Region', label: 'Region', type: 'string' },
    { field: 'Name', label: 'Name', type: 'string' }
];
}
onReset(e: Event): void {
    this.qryBldrObj?.reset();
}
onSetSqlRules(e: Event): void {

```

```

    this.qryBldrObj?.setRulesFromSql("Employee.ID = 0 AND Name.LastName LIKE ('%malan%') AND (Country.State.City LIKE ('U%') AND Country.Region LIKE ('%c') AND Country.Name IS NOT EMPTY)");
  }
  onSetRules(e:Event): void {
    this.qryBldrObj?.setRules((this as any).importRules);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Filtering in Angular Query builder component

Query Builder allows you to create or delete conditions and groups. You can use [showButtons](#) to enable/disable these buttons.

You can create or delete conditions by interacting through the user interface and methods.

- Use the [addRules](#), and [deleteRules](#) methods to create/delete conditions.
- Use [addGroups](#), and [deleteGroups](#) methods to create/delete groups.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
    [dataSource]="data" [rule]="importRules" [showButtons]="showButtons">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>

```



```

        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>
<button class="e-btn e-primary e-qb-button"
(click)="addRule()" >Add Rule</button>
<button class="e-btn e-primary e-qbr-button"
(click)="addGroup()" >Add Group</button>
<button class="e-btn e-primary e-qbr-button"
(click)="deleteGroup()" >Delete Group</button>`
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    public showButtons: Object = {ruleDelete: true, groupInsert: true,
groupDelete: true};
    values: any;
    ngOnInit(): void {
        this.data = employeeData;
        this.importRules = {
            'condition': 'and',
            'rules': [{
                'label': 'Employee ID',
                'field': 'EmployeeID',
                'type': 'number',
                'operator': 'equal',
                'value': 1
            },
            {
                'label': 'Title',
                'field': 'Title',
                'type': 'string',
                'operator': 'equal',
                'value': 'Sales Manager'
            }
        ]
    };
}
    addRule(): void {
        this.qryBldrObj!.addRules([{ 'label': 'City', 'field': 'City', 'type':
'string', 'operator': 'equal', 'value': 'US' }], 'group0');
    }
    addGroup(): void {
        this.qryBldrObj!.addGroups([{ 'condition': 'and', 'rules': [{ 'label':
'First Name', 'field': 'FirstName', 'type': 'string', 'operator':
'startswith', 'value': 'v' } ]}], 'group0');
    }
    deleteGroup(): void {
        this.qryBldrObj!.deleteGroups(['group1']);
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Template in Angular Query builder component

Templates allows users to define customized header and own user interface for columns.

Header Template

Header Template allows to define your own user interface for Header, which includes creating or deleting rules and groups and to customize the AND/OR condition and NOT condition options. To implement header template you can create the user interface using `ngTemplate` and assign the values when `requestType` is `header-template-create` in `actionBegin` event.

The `#headerTemplate` template variable identifies the `NgTemplate` content as the header.

In the following sample dropdown, splitbutton and button are used as the custom components in the header.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { Component, ViewChild, OnInit } from '@angular/core';
import { QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { ActionEventArgs, RuleModel } from '@syncfusion/ej2-querybuilder';
import { ItemModel, MenuEventArgs } from '@syncfusion/ej2-splitbuttons';
import { closest } from '@syncfusion/ej2-base';

@Component({
  imports: [

    QueryBuilderModule,
    CheckBoxModule,
    DropDownListModule,
    DropDownButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: `template-driven.html`
})
export class AppComponent implements OnInit {
  @ViewChild('querybuilder') qryBldrObj: QueryBuilderComponent | undefined;
  public ds: { [key: string]: Object }[] = [{ 'key': 'AND', 'value': 'and' }, { 'key': 'OR', 'value': 'or' }];
  public ddbitems?: ItemModel[];
  public importRules?: RuleModel;
  public actionArgs?: ActionEventArgs;
```

```

public deleteGroupBtn?: Element;
public fields?: Object;
ngOnInit(): void {
    this.importRules = {
        'condition': 'and', 'not': true,
        'rules': [{
            'label': 'Age',
            'field': 'Age',
            'type': 'number',
            'operator': 'equal',
            'value': 34
        },
        {
            'label': 'LastName',
            'field': 'LastName',
            'type': 'string',
            'operator': 'equal',
            'value': 'vinit'
        },
        {
            'condition': 'or',
            'rules': [{
                'label': 'Age',
                'field': 'Age',
                'type': 'number',
                'operator': 'equal',
                'value': 34
            }]
        }
    ];
    this.ddbitems = [
        {
            text: 'AddGroup',
            iconCss: 'e-icons e-add-icon e-addgroup'
        },
        {
            text: 'AddCondition',
            iconCss: 'e-icons e-add-icon e-addrule'
        }
    ];
    this.fields = { text: 'key', value: 'value' };
}
onChange(e: any): void {
    this.qryBldrObj!.notifyChange(e.checked, e.event.target, 'not');
}
conditionChange(e: any): void {
    this.qryBldrObj!.notifyChange(e.value, e.element, 'condition');
}
onSelect(event: MenuEventArgs): void {
    let addbtn: Element = closest(event.element, '.e-dropdown-popup'); let
ddbId: string = addbtn.id;
    let ddb: string[] = ddbId.split('_');
    if (event.item.text === 'AddGroup') {
        this.qryBldrObj!.addGroups([{condition: 'or', 'rules': [{}]}, not:
false]], ddb[1]);
    } else if (event.item.text === 'AddCondition') {
        this.qryBldrObj!.addRules([{}], ddb[1]);
    }
}

```

```

    }
    onClick(e: any): void {
        this.qryBldrObj!.deleteGroup(closest(e.target.offsetParent, '.e-group-container'));
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

TEMPLATE-DRIVEN.HTML

```

{% raw %}
<ejs-querybuilder id="querybuilder" #querybuilder width="100%" [rule] =
"importRules" enableNotCondition = true>
<e-columns>
<e-column field="EmployeeID" label="EmployeeID" type="number"></e-column>
<e-column field="FirstName" label="FirstName" type="string"></e-column>
<e-column field="LastName" label="LastName" type="string"></e-column>
<e-column field="Age" label="Age" type="number"></e-column>
<e-column field="City" label="City" type="string"></e-column>
<e-column field="Country" label="Country" type="string"></e-column>
</e-columns>
<ng-template #headerTemplate let-data>
<div class = "e-groupheader">
<button *ngIf="data.notCondition !== undefined" class='e-cb-wrapper'>
<ejs-checkbox id="{{data.ruleID}}_notOption" label='not'
[checked]='data.notCondition' (change)="onChange($event)">
</ejs-checkbox> </button>
<ejs-dropdownlist id="{{data.ruleID}}_cndtn" [dataSource]='ds'
[value]='data.condition' [fields]='fields' cssClass="e-custom-group-btn"
(change)="conditionChange($event)">
</ejs-dropdownlist>
<button ejs-dropdownbutton id="{{data.ruleID}}_addbtn" [items]='ddbitems'
cssClass= "e-round e-small e-caret-hide e-addrulegroup e-add-btn"
iconCss="e-icons e-add-icon" (select)="onSelect($event)"></button>
<button ejs-button *ngIf ="data.ruleID !== 'querybuilder_group0'" id=
'{{data.ruleID}}_dltbtn' class= "e-btn e-delete-btn e-lib e-small e-round e-
icon-btn" (click)="onClick($event)">
<span class = 'e-btn-icon e-icons e-delete-icon'></span>
</button>
</div>
</ng-template>
</ejs-querybuilder>
{% endraw %}

```

Column Template

Template allows you to define your own input widgets for columns. To implement [template](#), you can define the following functions

- **create**: Creates the custom component.
- **write**: Wire events for the custom component.
- **Destroy**: Destroy the custom component.

In the following sample, dropdown is used as the custom component in the PaymentMode column.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent, ColumnsModel, TemplateColumn }
from '@syncfusion/ej2-angular-querybuilder';
import { expenseData } from './datasource';
import { DropDownList, MultiSelect } from '@syncfusion/ej2-dropdowns';
import { getComponent, createElement } from '@syncfusion/ej2-base';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules" [columns]="filter">
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  @ViewChild('querybuilder')
  public qryBldrObj?: QueryBuilderComponent;
  public filter?: ColumnsModel[];
  public paymentTemplate?: TemplateColumn;
  public inOperators: string[] = ['in', 'notin'];
  ngOnInit(): void {
    this.data = expenseData;
    this.paymentTemplate = {
      create: () => {
        return createElement('input', { attrs: { 'type': 'text' } });
      },
      destroy: (args: { elementId: string }) => {
        let multiSelect: MultiSelect =
(getComponent(document.getElementById(args.elementId) as any, 'multiselect')
as MultiSelect);
        if (multiSelect) {
          multiSelect.destroy();
        }
        let dropdown: DropDownList =
(getComponent(document.getElementById(args.elementId) as any,
'dropdownlist') as DropDownList);
        if (dropdown) {
          dropdown.destroy();
        }
      }
    };
  }
}
```

```

    },
    write: (args: { elements: Element, values: string[] | string,
operator: string }) => {
        let ds: string[] = ['Cash', 'Debit Card', 'Credit Card', 'Net
Banking', 'Wallet'];
        if (this.inOperators.indexOf(args.operator) > -1) {
            let multiSelectObj: MultiSelect = new MultiSelect({
                dataSource: ds,
                value: args.values as string[],
                mode: 'CheckBox',
                placeholder: 'Select Transaction',
                change: (e: any) => {
                    this.qryBldrObj!.notifyChange(e.value, e.element);
                }
            });
            multiSelectObj.appendTo('#' + args.elements.id);
        } else {
            let dropDownObj: DropDownList = new DropDownList({
                dataSource: ds,
                value: args.values as string,
                change: (e: any) => {
                    this.qryBldrObj!.notifyChange(e.itemData.value,
e.element);
                }
            });
            dropDownObj.appendTo('#' + args.elements.id);
        }
    }
};

this.filter = [
    { field: 'Category', label: 'Category', type: 'string' },
    { field: 'PaymentMode', label: 'Payment Mode', type: 'string',
template: this.paymentTemplate },
    { field: 'TransactionType', label: 'Transaction Type', type:
'string' },
    { field: 'Description', label: 'Description', type: 'string' },
    { field: 'Date', label: 'Date', type: 'date' },
    { field: 'Amount', label: 'Amount', type: 'number' }
];

this.importRules = {
    'condition': 'and',
    'rules': [{
        'label': 'Payment Mode',
        'field': 'PaymentMode',
        'type': 'string',
        'operator': 'equal',
        'value': 'Cash'
    }]
};
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Using Template

The value template for a particular column can be specified using the content of the NgTemplate. The `#template` template variable identifies the NgTemplate content as the corresponding column.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { RuleModel } from '@syncfusion/ej2-querybuilder';
@Component({
  imports: [
    QueryBuilderModule,
    CheckBoxModule,
    DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: `template-driven.html`
})
export class AppComponent implements OnInit {
  @ViewChild('querybuilder') qryBldrObj: QueryBuilderComponent | undefined;
  public paymentDataSource: string[] = ['Cash', 'Debit Card', 'Credit Card', 'Net Banking'];
  public importRules?: RuleModel;
  public customOperators?: any;
  ngOnInit(): void {
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'Transaction Type',
        'field': 'TransactionType',
        'type': 'string',
        'operator': 'equal',
        'value': 'Expense'
      },
      {
        'label': 'Payment Mode',
        'field': 'PaymentMode',
        'type': 'string',
        'operator': 'equal',
        'value': 'Cash'
      }
    ]
  };
  this.customOperators = [
```

```

        {value: 'equal', key: 'Equal'},
        {value: 'notequal', key: 'Not Equal'}
    ];
}
transactionChange(e: any, ruleID: string): void {
    let elem: HTMLElement =
document.getElementById(ruleID)!.querySelector('.e-rule-value') as
HTMLElement;
    this.qryBldrObj!.notifyChange(e.checked === true ? 'Expense' : 'Income',
elem, 'value');
}
paymentChange(e: any, ruleID: string): void {
    let elem: HTMLElement =
document.getElementById(ruleID)!.querySelector('.e-rule-value') as
HTMLElement;
    this.qryBldrObj!.notifyChange(e.value as string, elem, 'value');
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Rule Template

Rule Template allows to define your own user interface for columns. To implement [ruleTemplate](#) you can create the user interface using `ngTemplate` and assign the values through `actionBegin` event.

The `#ruleTemplate` template variable identifies the `NgTemplate` content as the corresponding column.

In the following sample, dropdown and slider are used as the custom component and applied `greaterthanorequal` operator to `Age` column.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { QueryBuilderComponent } from '@syncfusion/ej2-angular-
querybuilder';
import { ActionEventArgs, RuleModel } from '@syncfusion/ej2-querybuilder';
import { compile } from '@syncfusion/ej2-base';
import { DataManager, Predicate, Query } from '@syncfusion/ej2-data';
import { employeeData } from './datasource';
@Component({
    imports: [
        QueryBuilderModule,
        SliderModule,

```



```

        DropDownListModule
    ],
    standalone: true,
    selector: 'app-root',
    templateUrl: `template-driven.html`
})
export class AppComponent implements OnInit {
    @ViewChild('querybuilder') qryBldrObj: QueryBuilderComponent | undefined;
    public importRules?: RuleModel;
    public rangeticks?: Object;
    ngOnInit(): void {
        this.importRules = {
            'condition': 'and',
            'rules': [{
                'label': 'Age',
                'field': 'Age',
                'type': 'number',
                'operator': 'greaterthanorequal',
                'value': 32
            }]
        };
        this.rangeticks = { placement: 'Before', largeStep: 5, smallStep: 1,
        showSmallTicks: true };
    }
    actionBegin(args: ActionEventArgs): void {
        if (args.requestType === 'template-initialize') {
            args.rule!.operator = 'greaterthanorequal';
            if (args.rule!.value === '') {
                args.rule!.value = 30;
            }
        }
    }
    fieldChange(e: any): void {
        this.qryBldrObj!.notifyChange(e.value, e.element, 'field');
    };
    valueChange(e: any, ruleID: string): void {
        let elem: HTMLElement = document.getElementById(ruleID) as HTMLElement;
        this.qryBldrObj!.notifyChange(e.value as Date, elem, 'value');
        this.refreshTable(this.qryBldrObj!.getRule(elem), ruleID);
    }
    viewDetails(ruleID: string): void {
        let ruleElem: HTMLElement = document.getElementById(ruleID) as
        HTMLElement;
        let element: HTMLElement = document.getElementById(ruleID + '_section')
        as HTMLElement;
        if (element.className.indexOf('e-hide') > -1) {
            this.refreshTable(this.qryBldrObj!.getRule(ruleElem), ruleID);
            element.className = element.className.replace('e-hide', '');
            document.getElementById(ruleID + '_option')!.querySelector('.e-
            content')!.textContent = 'Hide Details';
        } else {
            element.className += ' e-hide';
            document.getElementById(ruleID + '_option')!.querySelector('.e-
            content')!.textContent = 'View Details';
        }
    }
    refreshTable(rule: RuleModel, ruleID: string): void {

```

```

    let template: string =
    '<tr><td>${EmployeeID}</td><td>${FirstName}</td><td>${Age}</td></tr>';
    let compiledFunction: any = compile(template);
    let dataManagerQuery: Query =
    this.qryBldrObj!.getDataManagerQuery({condition: 'and', rules: [rule]});
    let dataManager: DataManager = new DataManager(employeeData);
    dataManager.defaultQuery = dataManagerQuery;
    let result: object[] = dataManager.executeLocal();
    let table: HTMLElement = document.getElementById(ruleID + '_datatable')
    as HTMLElement;
    if (table) {
        if (result.length) {
            table.style.display = 'block';
        } else {
            table.style.display = 'none';
        }
        table.querySelector('tbody')!.innerHTML = '';
        result.forEach((data) => {

        table.querySelector('tbody')!.appendChild(compiledFunction(data)[0].querySel
        ector('tr'));
        });
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

TEMPLATE-DRIVEN.HTML

```

<ejs-querybuilder id="querybuilder" #querybuilder width="100%" [rule] =
"importRules" (actionBegin)="actionBegin($event)">
    <e-columns>
        <e-column field="EmployeeID" label="Employee ID" type="number"></e-
column>
        <e-column field="FirstName" label="First Name" type="string"></e-
column>
        <e-column field="LastName" label="LastName" type="string"></e-
column>
        <e-column field="Age" label="Age" type="number">
            <ng-template #ruleTemplate let-data>
                <div class="e-rule e-rule-template">
                    <div class="e-rule-header">
                        <div class="e-rule-filter">
                            <ejs-dropdownlist (change)="fieldChange($event)"
[fields]="data.fields" [dataSource]="data.columns"
[value]="data.rule.field">
                                </ejs-dropdownlist>
                            </div>
                            <div *ngIf="data.rule.type === 'number'" class="e-
rule-value e-slide-val">

```

```

<ejs-slider [value]='data.rule.value'
[ticks]='rangeticks' min=30 max=50 id = "{{data.ruleID}}_valuekey0"
(change)="valueChange($event, data.ruleID)">
</ejs-slider>
</div>
<div class="e-rule-btn">
<button id="{{data.ruleID}}_option"
(click)="viewDetails(data.ruleID)" class="e-primary e-btn e-small">
<span class='e-content'>View Details</span>
</button>
<button class="e-remove-rule e-rule-delete e-css
e-btn e-small e-round">
<span class="e-btn-icon e-icons e-delete-
icon"></span>
</button>
</div>
</div>
<div id="{{data.ruleID}}_section" class="e-rule-content
e-hide">
<table id="{{data.ruleID}}_datatable" class='e-rule-
table e-hide'>
<thead>
<tr><th>EmployeeID</th><th>FirstName</th><th>Age</th></tr></thead>
<tbody></tbody>
</table>
</div>
</div>
</ng-template>
</e-column>
<e-column field="City" label="City" type="string"></e-column>
<e-column field="Country" label="Country" type="string"></e-column>
</e-columns>
</ejs-querybuilder>

```

Model binding in Angular Query builder component

Model binding allows to bind properties for the components used in field, operator, and value columns. To implement model binding, assign fieldModel, operatorModel, and valueModel properties in QueryBuilder.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DropDownListModule, MultiSelectModule } from '@syncfusion/ej2-
angular-dropdowns'
import { CheckBoxModule, RadioButtonModule } from '@syncfusion/ej2-angular-
buttons'
import { TextBoxModule, NumericTextBoxModule } from '@syncfusion/ej2-
angular-inputs'
import { DropDownButtonModule } from '@syncfusion/ej2-angular-splitbuttons'
import { DatePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild, OnInit } from '@angular/core';
import { QueryBuilderComponent } from '@syncfusion/ej2-angular-
querybuilder';
import { RuleModel } from '@syncfusion/ej2-querybuilder';

```

```

@Component({
  imports: [

    QueryBuilderModule,
    CheckBoxModule,
    DropDownListModule,
    DropDownButtonModule,
    DatePickerModule,
    TextBoxModule,
    NumericTextBoxModule,
    MultiSelectModule,
    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: `template-driven.html`
})
export class AppComponent implements OnInit {
  @ViewChild('querybuilder') qryBldrObj: QueryBuilderComponent | undefined;
  public importRules?: RuleModel;
  ngOnInit(): void {
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'Employee ID',
        'field': 'EmployeeID',
        'type': 'number',
        'operator': 'equal',
        'value': 1001
      }]
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Importing and Exporting in Angular Query builder component

Importing facilitates the viewing or editing of predefined conditions available in JSON, SQL, and MongoDB query formats, while exporting enables obtaining the created rules in the query builder as JSON, SQL, and MongoDB queries.

Importing

Importing enables users to bring predefined conditions into the system for viewing or editing, available in formats such as JSON, SQL, and MongoDB query. It facilitates the quick incorporation of pre-defined rules or parameters into workflows, streamlining the setup process by importing directly from external sources or saved configurations.

Importing from JSON Object

Importing from JSON enables users to bring predefined conditions encoded in JSON format into the system. This feature streamlines the process by providing a standardized format for importing data, ensuring compatibility, and ease of use

Initial rendering

To initially apply conditions, you can establish the [rule](#) by importing a structured JSON object and defining its properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
      'condition': 'or',
      'rules': [{
        'label': 'Category',
        'field': 'Category',
        'type': 'string',
        'operator': 'equal',
        'value': 'Laptop'
      ]
    }
  }
}
```

```

    },
    {
      'condition': 'and',
      'rules': [{
        'label': 'Status',
        'field': 'Status',
        'type': 'string',
        'operator': 'notequal',
        'value': 'Pending'
      },
      {
        'label': 'Task ID',
        'field': 'TaskID',
        'type': 'number',
        'operator': 'equal',
        'value': 5675
      }
    ]
  }
];
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Post rendering

You can set the conditions from structured JSON object through the [setRules](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
    [dataSource]="data">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
        type="number"></e-column>

```

```

        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>
<button class="e-btn e-primary e-qb-button"
(click)="setRules()" >Set Rules</button>`
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    ngOnInit(): void {
        this.data = hardwareData;
        this.importRules = {
            'condition': 'or',
            'rules': [{
                'label': 'Category',
                'field': 'Category',
                'type': 'string',
                'operator': 'equal',
                'value': 'Laptop'
            },
            {
                'condition': 'and',
                'rules': [{
                    'label': 'Status',
                    'field': 'Status',
                    'type': 'string',
                    'operator': 'notequal',
                    'value': 'Pending'
                },
                {
                    'label': 'Task ID',
                    'field': 'TaskID',
                    'type': 'number',
                    'operator': 'equal',
                    'value': 5675
                }
            ]
        }
    ];
    }
    setRules(): void {
        this.qryBldrObj!.setRules((this as any).importRules);
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Importing from SQL Query

Importing from SQL involves integrating predefined conditions or data stored in a SQL database into the Query Builder. This enables the direct integration of SQL queries, thereby improving workflow efficiency and data accuracy within the application. SQL importing supports various types, including Inline SQL, Parameter SQL, and Named Parameter SQL.

Importing from Inline SQL Query

Importing from Inline SQL involves integrating SQL queries directly into the Query Builder. This method streamlines the process by enabling users to input SQL statements directly into the application for analysis, manipulation, or further processing within the Query Builder. Conditions can be set from Inline SQL queries using the [setRulesFromSql](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="setRules()" >Set Rules</button>`
})
```



```
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    ngOnInit(): void {
        this.data = hardwareData;
    }
    setRules(): void {
        this.qryBldrObj!.setRulesFromSql("TaskID = 1 and Status LIKE ('Assigned%')");
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Importing from Parameter SQL Query

Importing from Parameter SQL involves integrating SQL queries with parameters directly into the Query Builder. This method allows users to input SQL statements containing parameters, which can be dynamically filled in during execution. It streamlines the process by enabling flexible and customizable querying within the application. Conditions can be set from Parameter SQL queries using the [setParameterizedSql](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
    imports: [
        QueryBuilderModule,
    ],
    standalone: true,
    selector: 'app-root',
    template: `<!-- To render Query Builder. -->
        <ejs-querybuilder #querybuilder class="row" width="70%"
        [dataSource]="data">
            <e-columns>
                <e-column field="TaskID" label="Task ID"
                type="number"></e-column>
                <e-column field="Name" label="Name" type="string"></e-column>
            </e-columns>
        </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    ngOnInit(): void {
        this.data = hardwareData;
    }
    setRules(): void {
        this.qryBldrObj!.setRulesFromSql("TaskID = 1 and Status LIKE ('Assigned%')");
    }
}
```

```

        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="setRules()" >Set Parameter SQL Rules</button>`
))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: any;
    ngOnInit(): void {
        this.data = hardwareData;
    }
    setRules(): void {
        this.qryBldrObj!.setParameterizedSql({ sql: '(Category IN (?,?) OR
TaskID IN (?,?))', params: ['Laptop', 'Others', 1, 2] });
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Importing from Named Parameter SQL Query

Importing from Named Parameter SQL involves integrating SQL queries with named parameters directly into the Query Builder. This method enables users to input SQL statements containing named parameters, providing flexibility and customization during execution. It streamlines the process by allowing dynamic parameter assignment within the application's query environment. Conditions can be set from Named Parameter SQL queries using the [setParameterizedNamedSql](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-
querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
imports: [

```

```

        QueryBuilderModule,
    ],
    standalone: true,
    selector: 'app-root',
    template: `<!-- To render Query Builder. -->
        <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data">
            <e-columns>
                <e-column field="TaskID" label="Task ID"
type="number"></e-column>
                <e-column field="Name" label="Name" type="string"></e-
column>
                <e-column field="Category" label="Category"
type="string"></e-column>
                <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
                <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
                <e-column field="Status" label="Status" type="string"></e-
column>
            </e-columns>
        </ejs-querybuilder>
        <button class="e-btn e-primary e-qb-button"
(click)="setRules()" >Set Named Parameter SQL Rules</button>`
    ))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: any;
    ngOnInit(): void {
        this.data = hardwareData;
    }
    setRules(): void {
        this.qryBldrObj!.setParameterizedNamedSql({ sql: '(Category IN
(:Category_1,:Category_2) OR TaskID IN (:TaskID_1,:TaskID_2))', params:
{"Category_1": "Laptop", "Category_2": "Others", "TaskID_1": 1, "TaskID_2":
2} });
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Importing from MongoDB Query

Importing from MongoDB Query involves integrating MongoDB queries directly into the Query Builder. This enables users to input MongoDB query statements directly into the application, allowing for seamless integration and manipulation of MongoDB data within the Query Builder environment. It streamlines the process by facilitating direct access to MongoDB data for analysis, filtering, and further

processing within the application. Conditions can be set from Named Parameter SQL queries using the [setMongoQuery](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="setRules()" >Set MongoDB Rules</button>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  @ViewChild('querybuilder')
  public qryBldrObj?: any;
  ngOnInit(): void {
    this.data = hardwareData;
  }
  setRules(): void {
    this.qryBldrObj!.setMongoQuery('{ "$and": [{"TaskID":1001},{
"$or": [{"Category":{"$regex":"Order"}}]}] }');
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Exporting

Exporting from the Query Builder allows users to preserve or store the created conditions. The defined conditions can be exported using various methods, including:

Exporting to JSON Object

You can extract the established conditions in the Query Builder and convert them into a structured JSON object format using the [getRules](#) method. This process enables users to save or transfer the conditions for further use or analysis in other applications or systems that support JSON data.

Exporting to SQL Query

Exporting to SQL involves converting the defined conditions within the Query Builder into SQL queries. This functionality allows users to generate SQL code representing the conditions set in the Query Builder, which can then be executed directly on a SQL database or used for further analysis and processing. SQL exporting supports various types, including Inline SQL, Parameter SQL, and Named Parameter SQL.

Exporting to Inline SQL Query

Exporting to Inline SQL Query entails embedding the defined conditions from the Query Builder directly into SQL statements within the exported code. This method ensures that the conditions are seamlessly integrated into the SQL query syntax, enabling straightforward execution or further processing within SQL database systems. This can be achieved using the [getSqlFromRules](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { hardwareData } from './datasource';
@Component({
  imports: [

    QueryBuilderModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
    [dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
        type="number"></e-column>
```

```

        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>
<button class="e-btn e-primary e-qb-button" (click)="getSql()"
>Get Sql</button>
<button class="e-btn e-primary e-qbr-button"
(click)="getJson()" >Get Json</button>
<ejs-dialog #dialog [header]="promptHeader"
[animationSettings]="animationSettings" [showCloseIcon]='showCloseIcon'
[height]="height" [width]="width" [visible]="hidden"></ejs-dialog>`
    ))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    @ViewChild('dialog')
    public Dialog?: DialogComponent;
    public animationSettings: Object = { effect: 'Zoom', duration: 400 };
    public showCloseIcon: Boolean = true;
    public hidden: Boolean = false;
    public width: string = '70%';
    public height: string = '80%';
    public promptHeader: string = 'Querybuilder Rule';
    ngOnInit(): void {
        this.data = hardwareData;
        this.importRules = {
            'condition': 'or',
            'rules': [{
                'label': 'Category',
                'field': 'Category',
                'type': 'string',
                'operator': 'equal',
                'value': 'Laptop'
            }]
        };
    }
    getSql(): void {
        this.Dialog!.content =
this.qryBldrObj!.getSqlFromRules(this.qryBldrObj!.getRules());
        this.Dialog!.show();
    }
    getJson(): void {
        this.Dialog!.content = '<pre>' + JSON.stringify({ condition:
this.qryBldrObj!.rule.condition, rules: this.qryBldrObj!.rule.rules }, null,
4) + '</pre>';
        this.Dialog!.show();
    }
}

```

```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Exporting to Parameter SQL Query

Exporting to Parameter SQL involves incorporating the defined conditions from the Query Builder into SQL queries with parameters. This method allows for dynamic value assignment during execution, enhancing flexibility and adaptability in query processing within SQL database. This can be accomplished using the [getParameterizedSql](#) method for exporting to Parameter SQL query.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
  imports: [

    QueryBuilderModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
    [dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
```

```

        <button class="e-btn e-primary e-qb-button" (click)="getSql()"
>Get Parameter Sql</button>
        <button class="e-btn e-primary e-qbr-button"
(click)="getJson()" >Get Json</button>
        <ejs-dialog #dialog [header]="promptHeader"
[animationSettings]="animationSettings" [showCloseIcon]='showCloseIcon'
[height]="height" [width]="width" [visible]="hidden"></ejs-dialog>`
    ))
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    @ViewChild('dialog')
    public Dialog?: any;
    public animationSettings: Object = { effect: 'Zoom', duration: 400 };
    public showCloseIcon: Boolean = true;
    public hidden: Boolean = false;
    public width: string = '70%';
    public height: string = '80%';
    public promptHeader: string = 'Querybuilder Rule';
    ngOnInit(): void {
        this.data = hardwareData;
        this.importRules = {
            'condition': 'or',
            'rules': [{
                'label': 'Category',
                'field': 'Category',
                'type': 'string',
                'operator': 'equal',
                'value': 'Laptop'
            }]
        };
    }
    getSql(): void {
        this.Dialog!.content =
JSON.stringify(this.qryBldrObj!.getParameterizedSql(this.qryBldrObj!.getRule
s()), null, 2);
        this.Dialog!.show();
    }
    getJson(): void {
        this.Dialog!.content = '<pre>' + JSON.stringify({ condition:
this.qryBldrObj!.rule.condition, rules: this.qryBldrObj!.rule.rules }, null,
4) + '</pre>';
        this.Dialog!.show();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```


Exporting to Named Parameter SQL Query

Exporting to Named Parameter SQL entails integrating the defined conditions from the Query Builder into SQL queries with named parameters. This method offers enhanced readability and flexibility during execution by using named placeholders for parameter values. Named Parameter SQL facilitates easier maintenance and modification of queries, making it convenient for dynamic parameter assignment within SQL database. This can be accomplished using the method [getParameterizedNamedSql](#) for exporting to Named Parameter SQL query.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { hardwareData } from '../datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
  imports: [

    QueryBuilderModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button" (click)="getSql()"
>Get Named Parameter Sql</button>
    <button class="e-btn e-primary e-qbr-button"
(click)="getJson()" >Get Json</button>
    <ejs-dialog #dialog [header]="promptHeader"
[animationSettings]="animationSettings" [showCloseIcon]='showCloseIcon'
[height]="height" [width]="width" [visible]="hidden"></ejs-dialog>`
  })
```

```

export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    @ViewChild('querybuilder')
    public qryBldrObj?: QueryBuilderComponent;
    @ViewChild('dialog')
    public Dialog?: any;
    public animationSettings: Object = { effect: 'Zoom', duration: 400 };
    public showCloseIcon: Boolean = true;
    public hidden: Boolean = false;
    public width: string = '70%';
    public height: string = '80%';
    public promptHeader: string = 'Querybuilder Rule';
    ngOnInit(): void {
        this.data = hardwareData;
        this.importRules = {
            'condition': 'or',
            'rules': [{
                'label': 'Category',
                'field': 'Category',
                'type': 'string',
                'operator': 'equal',
                'value': 'Laptop'
            }]
        };
    }
    getSql(): void {
        this.Dialog!.content =
        JSON.stringify(this.qryBldrObj!.getParameterizedNamedSql(this.qryBldrObj!.ge
tRules()), null, 2);
        this.Dialog!.show();
    }
    getJson(): void {
        this.Dialog!.content = '<pre>' + JSON.stringify({ condition:
this.qryBldrObj!.rule.condition, rules: this.qryBldrObj!.rule.rules }, null,
4) + '</pre>';
        this.Dialog!.show();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Exporting to MongoDB Query

Exporting to MongoDB Query involves converting the defined conditions within the Query Builder into MongoDB query syntax. This process allows users to generate MongoDB queries representing the conditions set in the Query Builder, which can then be executed directly on a MongoDB database or used for further analysis and processing. This can be accomplished using the [getMongoQuery](#) method for exporting to MongoDB query.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
  imports: [

    QueryBuilderModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data" [rule]="importRules">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button" (click)="getSql()"
>Get MongoDB</button>
    <button class="e-btn e-primary e-qbr-button"
(click)="getJson()" >Get Json</button>
    <ejs-dialog #dialog [header]="promptHeader"
[animationSettings]="animationSettings" [showCloseIcon]='showCloseIcon'
[height]="height" [width]="width" [visible]="hidden"></ejs-dialog>`
  })
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  @ViewChild('querybuilder')
  public qryBldrObj?: QueryBuilderComponent;
  @ViewChild('dialog')
  public Dialog?: any;
  public animationSettings: Object = { effect: 'Zoom', duration: 400 };
  public showCloseIcon: Boolean = true;
  public hidden: Boolean = false;
  public width: string = '70%';

```

```

public height: string = '80%';
public promptHeader: string = 'QueryBuilder Rule';
ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
        'condition': 'or',
        'rules': [{
            'label': 'Category',
            'field': 'Category',
            'type': 'string',
            'operator': 'equal',
            'value': 'Laptop'
        }]
    };
}

getSql(): void {
    this.Dialog!.content =
this.qryBldrObj!.getMongoQuery(this.qryBldrObj!.getRules());
    this.Dialog!.show();
}

getJSON(): void {
    this.Dialog!.content = '<pre>' + JSON.stringify({ condition:
this.qryBldrObj!.rule.condition, rules: this.qryBldrObj!.rule.rules }, null,
4) + '</pre>';
    this.Dialog!.show();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Lock Group/Rule in Angular Query builder component

The Query Builder provides the functionality to lock individual rules or entire groups. When a rule is locked, it prevents users from modifying its field, operator, and value, effectively disabling these components. Similarly, locking a group disables all elements contained within it. This feature offers users greater control over their query configurations, ensuring that specific rules or groups remain unchanged. Additionally, users can manage the visibility of locking buttons through the [showButtons](#) function, allowing for seamless control over the locking mechanism.

You can lock groups and rules by interacting through the user interface and methods.

- Use the [lockGroup](#) method to lock group.
- Use [lockRule](#) method to lock rule.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'

```

```

import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data" [rule]="importRules" [showButtons]="showButtons">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="lockGroup()" >Lock Group</button>
    <button class="e-btn e-primary e-qb-button"
(click)="lockRule()" >Lock Rule</button>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel = {
    'condition': 'and',
    'rules': [{
      'label': 'Task ID',
      'field': 'TaskID',
      'type': 'number',
      'operator': 'equal',
      'value': 1
    },
    {
      'label': 'Name',
      'field': 'Name',
      'type': 'string',
      'operator': 'equal',
      'value': 'Sales Manager'
    },
    {
      condition: "or", rules: [
        { 'label': 'Name',

```

```

        'field': 'Name',
        'type': 'string',
        'operator': 'equal',
        'value': 'Engineer' }
    ]
  }
];
};
public showButtons: Object = {ruleDelete: true, groupInsert: true,
groupDelete: true, lockGroup: false, lockRule: false};
@ViewChild('querybuilder')
public qryBldrObj?: QueryBuilderComponent;
ngOnInit(): void {
  this.data = hardwareData;
}
lockGroup(): void {
  this.qryBldrObj!.lockGroup("group0");
}
lockRule(): void {
  this.qryBldrObj!.lockRule("group0_rule0");
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Clone Group/Rule in Angular Query builder component

The Query Builder functionality extends to cloning both individual rules and entire groups. Utilizing the Clone options will generate an exact duplicate of a rule or group adjacent to the original one. This feature enables users to replicate complex query structures effortlessly. The [showButtons](#) function offers users the ability to toggle the visibility of these cloning buttons, providing convenient control over the cloning process within the Query Builder interface.

You can clone groups and rules by interacting through the user interface and methods.

- Use the [cloneGroup](#) method to clone group.
- Use [cloneRule](#) method to clone rule.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild, OnInit } from '@angular/core';
import { RuleModel, QueryBuilderComponent } from '@syncfusion/ej2-angular-querybuilder';
import { QueryLibrary } from '@syncfusion/ej2-querybuilder';
import { hardwareData } from './datasource';
QueryBuilderComponent.Inject(QueryLibrary);

```

```

@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder class="row" width="70%"
[dataSource]="data" [rule]="importRules" [showButtons]="showButtons">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>
    <button class="e-btn e-primary e-qb-button"
(click)="cloneGroup()" >Clone Group</button>
    <button class="e-btn e-primary e-qb-button"
(click)="cloneRule()" >Clone Rule</button>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  public showButtons: Object = {ruleDelete: true, groupInsert: true,
groupDelete: true, cloneGroup: false, cloneRule: false};
  @ViewChild('querybuilder')
  public qryBldrObj?: QueryBuilderComponent;
  ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'Task ID',
        'field': 'TaskID',
        'type': 'number',
        'operator': 'equal',
        'value': 1
      },
      {
        'label': 'Name',
        'field': 'Name',
        'type': 'string',
        'operator': 'equal',
        'value': 'Sales Manager'
      },
      {
        condition: "or", rules: [

```

```

        { 'label': 'Name',
          'field': 'Name',
          'type': 'string',
          'operator': 'equal',
          'value': 'Engineer' }
      ]
    }
  ];
}
cloneGroup(): void {
  this.qryBldrObj!.cloneGroup("querybuilder_group0",
"querybuilder_group1", 1);
}
cloneRule(): void {
  this.qryBldrObj!.cloneRule("querybuilder_group0_rule0",
"querybuilder_group0", 1);
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Global local in Angular Query builder component

The **Localization** library allows you to localize default text content of the Query Builder. The Query Builder component 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 Query Builder.

Locale key words	Text
-----	-----
AddGroup	Add Group
AddCondition	Add Condition
DeleteRule	Remove this condition
DeleteGroup	Delete group
Edit	EDIT
SelectField	Select a field
SelectOperator	Select operator
StartsWith	Starts With
EndsWith	Ends With
DoesNotStartWith	Does Not Start With

DoesNotEndWith	Does Not End With
Contains	Contains
DoesNotContain	Does Not Contain
Equal	Equal
NotEqual	Not Equal
LessThan	Less Than
LessThanOrEqual	Less Than Or Equal
GreaterThan	Greater Than
GreaterThanOrEqual	Greater Than Or Equal
Between	Between
NotBetween	Not Between
In	In
NotIn	Not In
Remove	REMOVE
ValidationMessage	This field is required
SummaryViewTitle	Summary View
OtherFields	Other Fields
AND	AND
OR	OR
SelectValue	Enter Value
IsEmpty	Is Empty
IsNotEmpty	Is Not Empty
IsNull	Is Null
IsNotNull	Is Not Null
True	True
False	False

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { hardwareData } from '../datasource';
setCulture('de-DE');
L10n.load({

```

```

    'de-DE': {
      'querybuilder': {
        'AddGroup': 'Gruppe hinzufügen',
        'AddCondition': 'Bedingung hinzufügen',
        'AddButton': "Gruppe/Bedingung hinzufügen",
        'DeleteRule': 'Entfernen Sie diesen Zustand',
        'DeleteGroup': 'Gruppe löschen',
        'Edit': 'BEARBEITEN',
        'SelectField': 'Wählen Sie ein Feld aus',
        'SelectOperator': 'Operator auswählen',
        'StartsWith': 'Beginnt mit',
        'EndsWith': 'Endet mit',
        'DoesNotStartWith': 'Beginnt nicht mit',
        'DoesNotEndWith': 'Endet nicht mit',
        'DoesNotContain': 'Beinhaltet nicht',
        'Contains': 'Enthält',
        'Equal': 'Gleich',
        'NotEqual': 'Nicht gleich',
        'LessThan': 'Weniger als',
        'LessThanOrEqual': 'Weniger als oder gleich',
        'GreaterThan': 'Größer als',
        'GreaterThanOrEqual': 'Größer als oder gleich',
        'Between': 'Zwischen',
        'NotBetween': 'Nicht zwischen',
        'In': 'Im',
        'NotIn': 'Nicht in',
        'Remove': 'LÖSCHEN',
        'ValidationMessage': 'Dieses Feld wird benötigt',
        'True': 'Wahr',
        'False': 'Falsch',
      }
    }
  });
@Component({
  imports: [
    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules" locale="de-DE">
    <e-columns>
      <e-column field="TaskID" label="Task ID"
type="number"></e-column>
      <e-column field="Name" label="Name" type="string"></e-
column>
      <e-column field="Category" label="Category"
type="string"></e-column>
      <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
      <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
      <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
  `;
})

```

```

        </ejs-querybuilder>`
    })
    export class AppComponent implements OnInit {
        public data?: Object[];
        public importRules?: RuleModel;
        ngOnInit(): void {
            this.data = hardwareData;
            this.importRules = {
                'condition': 'or',
                'rules': [{
                    'label': 'Category',
                    'field': 'Category',
                    'type': 'string',
                    'operator': 'equal',
                    'value': 'Laptop'
                }]
            };
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Style and appearance in Angular Query builder component

To modify the QueryBuilder appearance, you need to override the default CSS of QueryBuilder component. Please find the list of CSS classes and its corresponding section in QueryBuilder 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-group-header .e-btn|To customize the condition button in querybuilder |

|.e-group-body .e-rule-container|To customize the querybuilder rule container |

|.e-group-container .e-group-header .e-dropdown-btn|To customize the querybuilder Add group/condition button |

|.e-query-builder .e-group-header .e-deletegroup|To customize the querybuilder Delete group button |

|.e-query-builder .e-rule-field .e-rule-value-delete .e-rule-delete|To customize the querybuilder Delete condition button |

|.e-query-builder .e-rule-list > ::after,.e-query-builder .e-rule-list > ::before|To customize the querybuilder group joining line |

|.e-query-builder .e-rule-container.e-joined-rule|To customize the querybuilder condition joining line |

Accessibility in Angular Query Builder component

The Query Builder 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 Query Builder 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](#)

WAI-ARIA (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components.

The following list of ARIA attributes is used in Query Builder.

| Attributes | Purpose |

| --- | --- |

| **role** | Indicates the query builder component. |

Keyboard interaction

The Query Builder 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 Query Builder component.

| **Press** | **To do this** |

| --- | --- |

| **Tab / Shift + Tab** | To focus the next item in the rule. |

Ensuring accessibility

The Query Builder component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Query Builder component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Query Builder component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Display mode in Angular Query builder component

Display options allow you to view the Query Builder in Vertically or Horizontally. For this, you should use the [displayMode](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { employeeData } from '../datasource';
@Component({
  imports: [

    QueryBuilderModule,
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="40%"
    [dataSource]="data" [rule]="importRules" displayMode="Vertical">
    <e-columns>
```

```

        <e-column field="EmployeeID" label="Employee ID"
type="number" ></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
    </e-columns>
</ejs-querybuilder>`
    })
    export class AppComponent implements OnInit {
        public data?: Object[];
        public importRules?: RuleModel;
        values: any;
        ngOnInit(): void {
            this.data = employeeData;
            this.importRules = {
                'condition': 'and',
                'rules': [{
                    'label': 'EmployeeID',
                    'field': 'EmployeeID',
                    'type': 'number',
                    'operator': 'equal',
                    'value': 1001
                },
                {
                    'label': 'Title',
                    'field': 'Title',
                    'type': 'string',
                    'operator': 'equal',
                    'value': 'Sales Manager'
                }
            ]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

* The default view the query builder component is Horizontal.

* The default view the query builder component in Vertical.

Sort columns in Angular Query builder component

SortDirection allows you to sort the columns bounded to the Query Builder to view the columns by ascending or descending order. You should set the [sortDirection](#) property to sort the fields.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules" sortDirection="Descending">
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>
        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
      'condition': 'or',
      'rules': [{
        'label': 'Category',
        'field': 'Category',
        'type': 'string',
        'operator': 'equal',
        'value': 'Laptop'
      },
      {
        'condition': 'and',
        'rules': [{
          'label': 'Status',
```

```

        'field': 'Status',
        'type': 'string',
        'operator': 'notequal',
        'value': 'Pending'
    },
    {
        'label': 'Task ID',
        'field': 'TaskID',
        'type': 'number',
        'operator': 'equal',
        'value': 5675
    }
  ]
];
};
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Restrict groups in Angular Query builder component

You can restrict the condition set by defining the [maxGroupCount](#) property. By default, the value is 5. In the below demo, the `maxGroupCount` is set to 2.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
  imports: [
    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
    [dataSource]="data" [rule]="importRules" maxGroupCount=2>
      <e-columns>
        <e-column field="TaskID" label="Task ID"
type="number"></e-column>
        <e-column field="Name" label="Name" type="string"></e-
column>
        <e-column field="Category" label="Category"
type="string"></e-column>

```



```

        <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
        <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
        <e-column field="Status" label="Status" type="string"></e-
column>

    </e-columns>
</ejs-querybuilder>`
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public importRules?: RuleModel;
    ngOnInit(): void {
        this.data = hardwareData;
        this.importRules = {
            'condition': 'or',
            'rules': [{
                'label': 'Category',
                'field': 'Category',
                'type': 'string',
                'operator': 'equal',
                'value': 'Laptop'
            }]
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can use this property in the mobile mode to restrict the nested group creation.

State persistence in Angular Query builder component

State persistence allows you to maintain the current state 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 the Query Builder's [rule](#) object in the local storage when the [enablePersistence](#) is defined to true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
@Component({
    imports: [

        QueryBuilderModule
    ],

```

```

standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules" enablePersistence="true" >
    <e-columns>
      <e-column field="TaskID" label="Task ID"
type="number"></e-column>
      <e-column field="Name" label="Name" type="string"></e-
column>
      <e-column field="Category" label="Category"
type="string"></e-column>
      <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
      <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
      <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
  </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
      'condition': 'or',
      'rules': [{
        'label': 'Category',
        'field': 'Category',
        'type': 'string',
        'operator': 'equal',
        'value': 'Laptop'
      },
      {
        'condition': 'and',
        'rules': [{
          'label': 'Status',
          'field': 'Status',
          'type': 'string',
          'operator': 'notequal',
          'value': 'Pending'
        },
        {
          'label': 'Task ID',
          'field': 'TaskID',
          'type': 'number',
          'operator': 'equal',
          'value': 5675
        }
      ]
    }
  ];
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Rtl in Angular Query builder component

RTL provides an option to switch the text direction and layout of the Query Builder 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, set the [enableRtl](#) to true.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
import { hardwareData } from './datasource';
setCulture('ar-AE');
L10n.load({
  'ar-AE': {
    'querybuilder': {
      'AddGroup': 'إضافة مجموعة',
      'AddCondition': 'إضافة الشرط',
      'DeleteRule': 'أزل هذا الشرط',
      'DeleteGroup': 'حذف المجموعة',
      'Edit': 'تصحيح',
      'SelectField': 'اختر مجال',
      'SelectOperator': 'حدد المشغل',
      'StartsWith': 'ابدا ب',
      'EndsWith': 'ينتهي مع',
      'Contains': 'يحتوي على',
      'Equal': 'مساو',
      'NotEqual': 'ليس متساوي',
      'LessThan': 'أقل من',
      'LessThanOrEqual': 'اصغر من أو يساوي',
      'GreaterThan': 'أكبر من',
      'GreaterThanOrEqual': 'أكبر من أو يساوي',
      'Between': 'ما بين',
      'NotBetween': 'ليس بينهما',
      'In': 'في',
      'NotIn': 'ليس في',
      'Remove': 'إزالة',
      'ValidationMessage': 'هذه الخانة مطلوبه'
    }
  }
});
@Component({
  imports: [
    QueryBuilderModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="70%"
[dataSource]="data" [rule]="importRules" locale="ar-AE" enableRtl="true">
    <e-columns>
      <e-column field="TaskID" label="Task ID"
type="number"></e-column>
      <e-column field="Name" label="Name" type="string"></e-
column>
      <e-column field="Category" label="Category"
type="string"></e-column>
      <e-column field="SerialNo" label="SerialNo"
type="string"></e-column>
      <e-column field="InvoiceNo" label="InvoiceNo"
type="string"></e-column>
      <e-column field="Status" label="Status" type="string"></e-
column>
    </e-columns>
  </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  ngOnInit(): void {
    this.data = hardwareData;
    this.importRules = {
      'condition': 'or',
      'rules': [{
        'label': 'Category',
        'field': 'Category',
        'type': 'string',
        'operator': 'equal',
        'value': 'Laptop'
      }
    ]
  };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Summary view in Angular Query builder component

Summary view allows you to show or hide the filtered query. By default, the value is false. You can enable by setting the [summaryView](#) property to true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { QueryBuilderModule } from '@syncfusion/ej2-angular-querybuilder'

```

```

import { enableRipple } from '@syncfusion/ej2-base'
import { Component, OnInit } from '@angular/core';
import { employeeData } from './datasource';
import { RuleModel } from '@syncfusion/ej2-angular-querybuilder';
@Component({
  imports: [

    QueryBuilderModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Query Builder. -->
    <ejs-querybuilder #querybuilder width="30%"
[dataSource]="data" [rule]="importRules" summaryView="true">
      <e-columns>
        <e-column field="EmployeeID" label="Employee ID"
type="number"></e-column>
        <e-column field="FirstName" label="First Name"
type="string"></e-column>
        <e-column field="TitleOfCourtesy" label="Title Of
Courtesy" type="boolean" [values]="values"></e-column>
        <e-column field="Title" label="Title" type="string"></e-
column>
        <e-column field="HireDate" label="Hire Date" type="date"
format="dd/MM/yyyy"></e-column>
        <e-column field="Country" label="Country"
type="string"></e-column>
        <e-column field="City" label="City" type="string"></e-
column>
      </e-columns>
    </ejs-querybuilder>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public importRules?: RuleModel;
  public values: string[] = ['Mr.', 'Mrs.'];
  ngOnInit(): void {
    this.data = employeeData;
    this.importRules = {
      'condition': 'and',
      'rules': [{
        'label': 'EmployeeID',
        'field': 'EmployeeID',
        'type': 'number',
        'operator': 'notequal',
        'value': '5'
      }
    ],
    {
      'condition': 'or',
      'rules': [{
        'label': 'Title Of Courtesy',
        'field': 'TitleOfCourtesy',
        'type': 'string',
        'operator': 'equal',
        'value': 'Mr.'
      }
    ]
  },
  {

```

```

        'label': 'Country',
        'field': 'Country',
        'type': 'string',
        'operator': 'equal',
        'value': 'USA'
    },
    {
        'condition': 'and',
        'rules': [{
            'label': 'City',
            'field': 'City',
            'type': 'string',
            'operator': 'equal',
            'value': 'London'
        }]
    }
]
};
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

RadioButton

Getting started with Angular Radio button component

This section explains how to create a simple RadioButton, and demonstrate the basic usage of the RadioButton module in an Angular environment.

Dependencies

The following list of dependencies are required to use the RadioButton module in your application.

`typescript

|-- @syncfusion/ej2-angular-buttons

|-- @syncfusion/ej2-angular-base

|-- @syncfusion/ej2-base

|-- @syncfusion/ej2-buttons

,

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

,

npm install -g @angular/cli

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
ng new my-app  
cd my-app
```

Installing Syncfusion RadioButton package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(`>=20.2.36`) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-buttons](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-buttons --save
```

Angular compatibility compiled package(`ngcc`)

For Angular version below 12, you can use the legacy (`ngcc`) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-buttons@ngcc](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-buttons@ngcc --save
```

To mention the `ngcc` package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash  
@syncfusion/ej2-angular-buttons:"20.2.38-ngcc"
```

Note: If the `ngcc` tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding RadioButton module

Import RadioButton module into Angular application(app.module.ts) from the package

`@syncfusion/ej2-angular-buttons`.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Imported Syncfusion radiobutton module from buttons package.
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, RadioButtonModule ], // Registering EJ2 RadioButton Module.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding Syncfusion RadioButton component

Modify the template in `app.component.ts` file to render the RadioButton component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To Render RadioButton. -->
<ejs-radiobutton label="Default"></ejs-radiobutton>`
})
export class AppComponent { }
```

Adding CSS reference

Add RadioButton component's styles as given below in `style.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
```


Running the application

Run the application in the browser using the following command:

`

ng serve

`

The below example shows a basic RadioButton component,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ul>
      <!-- To Render RadioButton. -->
      <li><ejs-radiobutton label="Option 1" name="default"></ejs-
radiobutton></li>
      <li><ejs-radiobutton label="Option 2" name="default"
checked="true"></ejs-radiobutton></li>
    </ul>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Change the RadioButton state

The Essential JS 2 RadioButton contains 2 different states visually, they are as follows:

- Checked
- Unchecked

The RadioButton [checked](#) property is used to handle the checked and unchecked state.

In the checked state an inner circle will be added to the visualization of RadioButton.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ul>
      <!-- checked state. -->
      <li><ejs-radiobutton label="Option 1" name="state"
checked="true"></ejs-radiobutton></li>
      <!-- unchecked state. -->
      <li><ejs-radiobutton label="Option 2" name="state"></ejs-
radiobutton></li>
    </ul>
  </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Label and size in Angular Radio button component

This section explains the different sizes and labels.

Label

RadioButton caption can be defined by using the [label](#) property. This reduces the manual addition of label for RadioButton. You can customize the label position before or after the RadioButton through the [labelPosition](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
```

```

        <ul>
        <!-- Label position - Left. -->
        <li><ejs-radiobutton label="Left Side Label" name="position"
labelPosition="Before"></ejs-radiobutton></li>
        <!-- Label position - Right. -->
        <li><ejs-radiobutton label="Right Side Label" name="position"
checked="true"></ejs-radiobutton></li>
        </ul>
        </div>`
    ))
    export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Size

The different RadioButton size are default and small. To reduce the size of the default RadioButton to small, set the [cssClass](#) property to `e-small`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ul>
    <!-- Small RadioButton. -->
    <li><ejs-radiobutton label="Small" name="small"
checked="true" cssClass="e-small"></ejs-radiobutton></li>
    <!-- Default RadioButton. -->
    <li><ejs-radiobutton label="Default" name="small"></ejs-
radiobutton></li>
    </ul>
    </div>`
  })
  export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

See Also

- [How to customize the RadioButton appearance](#)

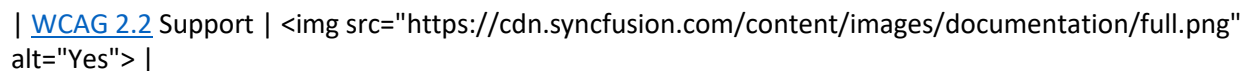
Accessibility in Angular Radio button component

The Radio 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 Radio button component is outlined below.

| Accessibility Criteria | Compatibility |

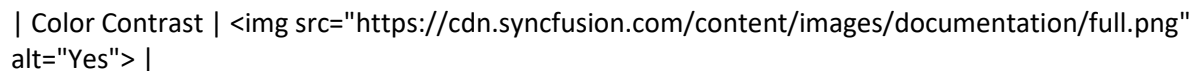
| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes"> |

| [Section 508](#) Support |  alt="Yes"> |

| Screen Reader Support |  alt="Yes"> |

| Right-To-Left Support |  alt="Yes"> |

| Color Contrast |  alt="Yes"> |

| Mobile Device Support |  alt="Yes"> |

| Keyboard Navigation Support |  alt="Yes"> |

| [Accessibility Checker](#) Validation |  alt="Yes"> |

| [Axe-core](#) Accessibility Validation |  alt="Yes"> |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> alt="Yes"> - 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 Radio button component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Radio button component:

| Attributes | Purpose |

| --- | --- |

| **aria-disabled** | Indicates that the element is perceivable but disabled, so it is not editable or otherwise operable. |

Keyboard interaction

The Radio 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 Radio button component.

| Press | To do this |

| --- | --- |

| UP/Left arrow | Move and select the previous options. |

| Down/Right arrow | Move and select the next options. |

Ensuring accessibility

The Radio button component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Radio button component is shown in the following sample. Open the [sample](https://ej2.syncfusion.com/accessibility/Radio button.html) in a new window to evaluate the accessibility of the Radio button component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Customize radiobutton appearance in Angular Radio button component

You can customize the appearance of the RadioButton component by 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 RadioButton is customized through the custom classes to create primary, success, warning, danger, and info type of radio button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```

```

import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // To customize RadioButton appearance
  template: `<div class="e-section-control">
    <ul>
      <!-- Refer the 'e-primary' class details in 'style.css'. -->
      <li><ejs-radiobutton label="Primary" name="custom"
cssClass="e-primary"></ejs-radiobutton></li>
      <!-- Refer the 'e-success' class details in 'style.css'. -->
      <li><ejs-radiobutton label="Success" name="custom"
cssClass="e-success"></ejs-radiobutton></li>
      <!-- Refer the 'e-info' class details in 'style.css'. -->
      <li><ejs-radiobutton label="Info" name="custom" cssClass="e-
info" checked="true"></ejs-radiobutton></li>
      <!-- Refer the 'e-warning' class details in 'style.css'. -->
      <li><ejs-radiobutton label="Warning" name="custom"
cssClass="e-warning"></ejs-radiobutton></li>
      <!-- Refer the 'e-danger' class details in 'style.css'. -->
      <li><ejs-radiobutton label="Danger" name="custom"
cssClass="e-danger"></ejs-radiobutton></li>
    </ul>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Name and value in form submit in Angular Radio button component

The [name](#) attribute of the RadioButton is used to group RadioButton. When the RadioButton are grouped in form, the checked item [value](#) attribute will be post to server on form submit

that can be retrieved through the name. The disabled and unchecked RadioButton value will not be sent to the server on form submit.

In the following code snippet, Credit and Debit card is in the checked state. Now, the value that is in checked state will be sent on form submit.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ButtonModule, RadioButtonModule } from '@syncfusion/ej2-angular-
buttons'
import { enableRipple } from '@syncfusion/ej2-base'

```

```

import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
import { NgForm } from '@angular/forms';
@Component({
  imports: [

    RadioButtonModule,
    ButtonModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // Name and Value attribute in form submit.
  template: ` <div class="e-section-control">
    <form #form="ngForm" (ngSubmit)="submitForm(form)">
    <ul>
      <li><ejs-radiobutton name="payment"
[ (ngModel) ]="selectedOption" value="credit/debit" label="Credit / Debit
card" checked="true"></ejs-radiobutton></li>
      <li><ejs-radiobutton name="payment"
[ (ngModel) ]="selectedOption" value="netbanking" label="Net Banking"></ejs-
radiobutton></li>
      <li><ejs-radiobutton name="payment"
[ (ngModel) ]="selectedOption" value="cashondelivery" label="Cask On
Delivery"></ejs-radiobutton></li>
      <li><ejs-radiobutton name="payment"
[ (ngModel) ]="selectedOption" value="others" label="Others"></ejs-
radiobutton></li>
      <li><button ejs-button
[isPrimary]="true">Submit</button></li>
    </ul>
  </form>
</div>`
})
export class AppComponent {
  public selectedOption: string = 'credit/debit';
  submitForm(form: NgForm): void {
    console.log(form.value);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Right to left in Angular Radio button component

RadioButton 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 RadioButton component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // To customize RadioButton appearance
  template: `<div class="e-section-control">
    <ul>
      <li><ejs-radiobutton label="Option 1" name="default"
enableRtl="true" checked="true"></ejs-radiobutton></li>
      <li><ejs-radiobutton label="Option 2" name="default"
enableRtl="true"></ejs-radiobutton></li>
    </ul>
  </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set the disabled state in Angular Radio button component

RadioButton component can be enabled/disabled by giving [disabled](#) property. To disable RadioButton component, the [disabled](#) property can be set as `true`.

The following example illustrates how to disable a radio button and the selected one is displayed using [change](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { ChangeEventArgs, RadioButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // To customize RadioButton appearance
  template: `<div class="e-section-control">

```



```

        <ul>
          <li>
            <div id="text">Selected : Option 1</div>
          </li>
          <li>
            <ejs-radiobutton #radio1 label="Option 1"
name="default" checked="true" (change)='changeOption1($event)'></ejs-
radiobutton>
          </li>
          <li>
            <ejs-radiobutton #radio2 label="Option 2"
name="default" disabled="true" (change)='changeOption2($event)'></ejs-
radiobutton>
          </li>
          <li>
            <ejs-radiobutton #radio3 label="Option 3"
name="default" (change)='changeOption3($event)'></ejs-radiobutton>
          </li>
        </ul>
      </div>`
    })
    export class AppComponent {
      @ViewChild('radio1')
      public radio1?: RadioButtonComponent;
      @ViewChild('radio2')
      public radio2?: RadioButtonComponent;
      @ViewChild('radio3')
      public radio3?: RadioButtonComponent;
      public changeOption1 (args: ChangeEventArgs) {
        document.getElementById('text')!.innerText = 'Selected : ' +
this.radio1!.label;
      }
      public changeOption2 (args: ChangeEventArgs) {
        document.getElementById('text')!.innerText = 'Selected : ' +
this.radio2!.label;
      }
      public changeOption3 (args: ChangeEventArgs) {
        document.getElementById('text')!.innerText = 'Selected : ' +
this.radio3!.label;
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Two way binding using radiobutton in Angular Radio button component

In the following example, two-way binding for RadioButton is illustrated with DropDownList component. The steps to achieve two-way binding in RadioButton are as follows,

- Initialize RadioButton component and bind the checked value using ngModel as in the below code using "banana in a box" syntax,

`typescript

```
<ejs-radiobutton [label]='payment' [value]="payment" name="payment" [(ngModel)]="value"></ejs-radiobutton>
```

,

- Initialize DropDownList component and assign the [value](#) property value like the below code,

`typescript

```
<ejs-dropdownlist [dataSource]='paymentMethod' [(value)]="value" ></ejs-dropdownlist>
```

,

- Now, the changes made in RadioButton will reflect in DropDownList (i.e. Selected option in radio button will be reflected in DropDownList) and vice versa.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { enableRipple } from '@syncfusion/ej2-base'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [

    RadioButtonModule,
    FormsModule,
    DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  // To customize RadioButton appearance
  template: ` <div class="e-section-control">
    <div class="radioButton-control">
      <h4>Select a payment method</h4>
      <div class="row" *ngFor="let payment of paymentMethod">
        <ejs-radiobutton [label]='payment' [value]="payment"
name="payment" [(ngModel)]="value"></ejs-radiobutton>
      </div>
    </div>
    <div class="dropDownList-control">
      <h4>Payment Method</h4>
      <ejs-dropdownlist [dataSource]='paymentMethod'
[ (value) ]="value" ></ejs-dropdownlist>
    </div>
  </div>`
})
```

```
export class AppComponent {
    public paymentMethod: string[] = ['Credit card', 'Debit card', 'Net Banking', 'Other Wallets'];
    public value:string = "Credit card";
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Ej1 api migration in Angular Radio button component

This article describes the API migration process of RadioButton component from Essential JS 1 to Essential JS 2.

Properties

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Label | **Property:** *text*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" /> | **Property:** *label*

 <ejs-radiobutton id="radio" label="RadioButton"></ejs-radiobutton> |

| Checked state | **Property:** *checked*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [checked]="true" /> | **Property:** *checked*

 <ejs-radiobutton id="radio" label="RadioButton" [checked]="true"></ejs-radiobutton> |

| Adding custom css class | **Property:** *cssClass*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" cssClass="custom-class" /> | **Property:** *cssClass*

 <ejs-radiobutton id="radio" label="RadioButton" cssClass="custom-class"></ejs-radiobutton> |

| Disabled state | **Property:** *enabled*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [enabled]="false" /> | **Property:** *disabled*

 <ejs-radiobutton id="radio" label="RadioButton" [disabled]="true"></ejs-radiobutton> |

| State persistence | **Property:** *enablePersistence*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [enablePersistence]="true" /> | **Property:** *enablePersistence*

 <ejs-radiobutton id="radio" label="RadioButton" [enablePersistence]="true"></ejs-radiobutton> |

| RTL | **Property:** *enableRTL*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [enableRTL]="true" /> | **Property:** *enableRtl*

 <ejs-radiobutton id="radio" label="RadioButton" [enableRtl]="true"></ejs-radiobutton> |

| HTML Attributes | **Property:** *htmlAttributes*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [htmlAttributes]="attributes" /> | Not applicable |

| Id property | **Property:** *id*

 <input type="radio" ej-radiobutton id="sync" /> | Not applicable |

| Prefix value of Id | **Property:** *idPrefix*

 <input type="radio" ej-radiobutton idPrefix="ng" /> | Not applicable |

| Name attribute | **Property:** *name*

 <input type="radio" ej-radiobutton id="radio" name="gender" [checked]="true" text="Male" /> | **Property:** *name*

 <ejs-radiobutton id="radio" name="gender" [checked]="true" label="Male"></ejs-radiobutton> |

| Value attribute | **Property:** *value*

 <input type="radio" ej-radiobutton id="radio" name="gender" [checked]="true" value="male" text="Male" /> | **Property:** *value*

 <ejs-radiobutton id="radio" name="gender" [checked]="true" value="male" label="Male"></ejs-radiobutton> |

| Size | **Property:** *size*

 <input type="radio" ej-radiobutton id="radio" size="small" text="RadioButton" /> | **Property:** *cssClass*

 <ejs-radiobutton id="radio" label="RadioButton" cssClass="e-small"></ejs-radiobutton> |

| Validation rules | **Property:** *validationRules*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [validationRules]="rules" /> | Not applicable |

| Validation message | **Property:** *validationMessage*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" [validationRules]="rules" [validationMessage]="message" /> | Not applicable |

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Destroy | **Method:** *destroy*

 <input #radio type="radio" ej-radiobutton id="radio" text="RadioButton" />
 @ViewChild('radio')
 public radioButton: RadioButtonComponent;
 this.radioButton.destroy(); | **Method:** *destroy*

 <ejs-radiobutton #radio id="radio" label="RadioButton"></ejs-radiobutton>
 @ViewChild('radio')
 public radioButton: RadioButtonComponent;
 this.radioButton.destroy(); |

| Disable the RadioButton | **Method:** *disable*

 <input #radio type="radio" ej-radiobutton id="radio" text="RadioButton" />
 @ViewChild('radio')
 public radioButton: RadioButtonComponent;
 this.radioButton.disable(); | Not applicable |

| Enable the RadioButton | **Method:** *enable*

 <input #radio type="radio" ej-radiobutton id="radio" text="RadioButton" />
 @ViewChild('radio')
 public radioButton: RadioButtonComponent;
 this.radioButton.enable(); | Not applicable |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| BeforeChange Event | **Event:** *beforeChange*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" (beforeChange)="beforeChange(\$event)" />
 beforeChange(args) {
 / code block */
 } | Not applicable |

| Change Event | **Event:** *change*

 <input type="radio" ej-radiobutton id="radio" text="RadioButton" (change)="change(\$event)" />
 change(args) {


```
</code></pre>

```

```
</code></pre>

```

```
</code></pre>

```

Range Navigator

Getting started with Angular Range navigator component

This section explains you the steps required to create a simple RangeNavigator and demonstrate the basic usage of the RangeNavigator component in an Angular environment.

Setup Angular Environment

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

To install Angular CLI use the following command.

```
`bash
npm install -g @angular/cli
`
```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
ng new my-app
cd my-app
`
```

Installing Syncfusion Chart package

Syncfusion packages are distributed in npm as [@syncfusion](#) scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(>=20.2.36) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-charts](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-charts --save
`
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the ngcc package use the below.

Add [@syncfusion/ej2-angular-charts@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-charts@ngcc --save
`
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-charts:"20.2.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Registering RangeNavigator Module

Import Chart module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-charts` [`src/app/app.module.ts`].

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the RangeNavigatorModule for the RangeNavigator component
import { RangeNavigatorModule } from '@syncfusion/ej2-angular-charts';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of RangeNavigatorModule into NgModule
  imports: [ BrowserModule, RangeNavigatorModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

- Modify the template in `app.component.ts` file to render the `ej2-angular-charts` component

[src/app/app.component.ts].

```
`javascript
```

```
import { Component, ViewEncapsulation } from '@angular/core';
```

```
@Component({
```

```
  selector: 'app-container',
```

```
  // specifies the template string for the RangeNavigator component
```

```
  template: <ejs-rangenavigator id="rn-container"></ejs-rangenavigator>,
```

```
  encapsulation: ViewEncapsulation.None
```

```
})
```

```
export class AppComponent { }
```

```
,
```

```
<!-- markdownlint-disable MD033 -->
```

Now use the `<code>app-container</code>`

```
,
```

```
<app-container></app-container>
```

```
,
```

- Now run the application in the browser using the below command.

```
,
```

```
npm start
```

```
,
```

Module Injection

To create range navigator with additional features, inject the required modules. The following modules are used to extend rangenavigator's basic functionality.

- `AreaSeriesService` - Inject this module to use area series.
- `DateTimeService` - Inject this module to use date time axis.
- `RangeTooltipService` - Inject this module to show the tooltip.

These modules should be injected to the provider section as follows,

```
`javascript
```

```
import { NgModule } from '@angular/core';
```

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
```

```
import { RangeNavigatorComponent } from '@syncfusion/ej2-angular-charts';

import { AreaSeriesService, DateTimeService, RangeTooltipService } from '@syncfusion/ej2-angular-charts';

@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [AppComponent, RangeNavigatorComponent],
  bootstrap: [AppComponent],
  providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ]
})
`
```

Populate Range Navigator with Data

Now, we are going to provide data to the range navigator. Add a series object to the range navigator by using `series` property. Now map the field names `x` and `y` 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 Datetime data, set the `valueType` as `DateTime`. By default, the axis `valueType` is `Numeric`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
```



```

public chartData?: Object[];
ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Note: Get data from [here](#).

The sample should look like our [default](#), don't worry about the gradient color, let it takes the default color.

Enable Tooltip

The tooltip is useful to show the selected data. You can enable tooltip by setting the enable property as true in tooltip object and by injecting RangeTooltipService module into the @NgModule.providers.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts';
import { AreaSeriesService, DateTimeService, RangeTooltipService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { datasrc } from './data';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [AreaSeriesService, DateTimeService, RangeTooltipService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [tooltip]='tooltip'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
  }
}

```

```

        this.tooltip = { enable: true, displayMode: 'Always' };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Selecting range in Angular Range navigator component

The Range Selector's left and right thumbs are used to indicate the selected range in the large collection of data. A range can be selected in the following ways:

- By dragging the thumbs.
- By tapping on the labels.
- By setting the start and the end through the [value](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, RangeTooltipService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { datasrc } from './datasource';
import {
    DateTime, AreaSeries, IChangedEventArgs, Chart
} from '@syncfusion/ej2-charts';
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [tooltip]='tooltip'
[labelFormat]='labelFormat' (changed)='changed($event)'>
        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>
    <div align="center">
        <ejs-chart #chart style='display:block;' id='chart'
[primaryXAxis]='primaryXAxis'>
            <e-series-collection>
                <e-series [dataSource]='chartData' type='Area' xName='x'
yName='y' width='2'>

```

```

        </e-series>
    </e-series-collection>
</ejs-chart>
</div>`
}))
export class AppComponent implements OnInit {
    public value?: Object[];
    public chartData?: Object[];
    public tooltip?: Object[];
    public labelFormat?: string;
    public primaryXAxis?: Object;
    @ViewChild('chart')
    public Chart?: Chart;
    public changed(args: IChangedEventArgs): void {
        (this.Chart as Chart).primaryXAxis.zoomFactor = args.zoomFactor;
        (this.Chart as Chart).primaryXAxis.zoomPosition = args.zoomPosition;
        (this.Chart as Chart).dataBind();
    }
    ngOnInit(): void {
        this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
        this.chartData = datasrc;
        this.tooltip = [{ enable: true, displayMode: 'Always' }];
        this.labelFormat = 'MMM-yy';
        this.primaryXAxis = {
            valueType: 'DateTime',
        };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Lightweight in Angular Range navigator component

By default, when the [dataSource](#) for [series](#) is empty, a lightweight Range Selector will be shown without Chart.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { GetDateTimeData } from './datasource'
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService ],
})

```

```

standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [labelFormat]='labelFormat' [dataSource]='chartData'
xName='x' yName='y'></ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public tooltip?: Object[];
  public labelFormat?: string;
  ngOnInit(): void {
    this.value = [new Date(2018, 4, 1), new Date(2018, 8, 1)];
    this.chartData = GetDateTimeData(new Date(2018, 0, 1), new Date(2019,
0, 1));
    this.labelFormat = 'MMM';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [Period Selector](#)

Series types in Angular Range navigator component

To render the data, the Range Selector supports three types of series.

<!-- markdownlint-disable MD036 -->

Line

<!-- markdownlint-disable MD036 -->

To render a line series, use series [type](#) as **Line** and inject the **LineSeriesService** into the **@NgModule.providers**. By default, the line series is rendered in the Range Selector .

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { LineSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],

```

```

providers: [ LineSeriesService, DateTimeService ],
standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Line'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public tooltip?: Object[];
  public labelFormat?: string;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.labelFormat = 'MMM-yy';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Area

To render an area series, use series [type](#) as **Area** and inject **AreaSeriesService** into the **@NgModule.providers**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value'>
    <e-rangenavigator-series-collection>

```

```

        <e-rangenavigator-series [dataSource]='chartData' type='Area'
        xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public value?: Object[];
        public chartData?: Object[];
        public tooltip?: Object[];
        public labelFormat?: string;
        ngOnInit(): void {
            this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
            this.chartData = datasrc;
            this.labelFormat = 'MMM-yy';
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

StepLine

To render a Step line series, use series [Type](#) as **Step Line** and inject **StepLineSeries** into the **@NgModule.providers**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource';
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ StepLineSeriesService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" [value]='value'>
        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData'
            type='StepLine' xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
        </ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public value?: Object[];
    }
}

```

```

public chartData?: Object[];
public tooltip?: Object[];
public labelFormat?: string;
ngOnInit(): void {
    this.value = [12, 30];
    this.chartData = datasrc;
    this.labelFormat = 'MMM-yy';
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

<!-- markdownlint-disable MD036 -->

Axis in Angular Range navigator component

Numeric

The numeric scale is used to represent the numeric values of data in a Range Selector. By default, the [valueType](#) of a Range Selector is **Double**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { double } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" [value]='value'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  ngOnInit(): void {
    this.value = [12, 30];
    this.chartData = double;
  }
}

```

```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Range

The minimum and the maximum of the scale will be calculated automatically based on the provided data. It can be customized by using the [minimum](#), [maximum](#), and [interval](#) properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" [value]='value'
[minimum]='minimum' [maximum]='maximum' [interval]='interval'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='xData' yName='yData' width=2>
        </e-rangenavigator-series>
      </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public minimum?: number;
  public maximum?: number;
  public interval?: number;
  ngOnInit(): void {
    this.value = [60,100];
    this.chartData = [
      { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
      { xData: 30, yData: 34 }, { xData: 40, yData: 32 },
      { xData: 50, yData: 40 }, { xData: 60, yData: 30 },
      { xData: 70, yData: 4 }, { xData: 80, yData: 22 }
    ],
  },
}
```



```

        { xData: 90, yData: 30 }, { xData: 100, yData: 43
    },
        { xData: 110, yData: 60 }, { xData: 120, yData:
33 },
        { xData: 130, yData: 40 }, { xData: 140, yData:
29 },
        { xData: 150, yData: 10 }, { xData: 160, yData:
16 },
    ];
    this.minimum= 10;
    this.maximum= 160;
    this.interval= 10;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label Format

The numeric labels can be formatted using the [labelFormat](#) property and it supports all the globalized formats.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { double } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" [value]='value'
labelFormat='n1'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
        </e-rangenavigator-series>
      </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  ngOnInit(): void {

```

```

        this.value = [12, 30];
        this.chartData = double;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

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

The Range Selector also supports the Custom Label formats using the placeholders such as **{value}\$**, in which the value represents the axis label, e.g. 20\$.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts';
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { double } from './datasource';
@Component({

```

```

imports: [
    ChartModule, RangeNavigatorModule
],
providers: [ StepLineSeriesService ],
standalone: true,
selector: 'app-container',
template: `<ejs-rangenavigator id="rn-container" [value]='value'
labelFormat='{value}$'>
    <e-rangenavigator-series-collection>
        <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
))
export class AppComponent implements OnInit {
    public value?: Object[];
    public chartData?: Object[];
    ngOnInit(): void {
        this.value = [12, 30];
        this.chartData = double;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Logarithmic Axis

The Logarithmic supports the logarithmic scale, and it is used to visualize the data when the Range Selector has numerical values in both the lower (e.g.: 10⁻⁶) and the higher (e.g.: 10⁶) orders of the magnitude.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService, LogarithmicService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ StepLineSeriesService, LogarithmicService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" [value]='value'
valueType='Logarithmic' [interval]='interval'>
        <e-rangenavigator-series-collection>

```

```

        <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public value?: Object[];
        public chartData?: Object[];
        public interval?: number;
        ngOnInit(): void {
            this.value = [4, 6];
            this.chartData = [];
            for (let i = 0; i < 100; i++) {
                this.chartData.push({
                    x: Math.pow(10, i * 0.1),
                    y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
                });
            }
            this.interval = 1;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

To use logarithmic scale, inject the `LogarithmicService` into the `@NgModule.providers`, and then set the `valueType` to `Logarithmic`.

Range

The minimum and the maximum of the Range Selector will be calculated automatically based on the provided data. It can be customized by using the [minimum](#), [maximum](#), and [interval](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts';
import { StepLineSeriesService, LogarithmicService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ StepLineSeriesService, LogarithmicService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" [value]='value'
valueType='Logarithmic' [interval]='interval'>

```

```

        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public value?: Object[];
        public chartData?: Object[];
        public interval?: number;
        ngOnInit(): void {
            this.value = [4, 6];
            this.chartData = [];
            for (let i = 0; i < 100; i++) {
                this.chartData.push({
                    x: Math.pow(10, i * 0.1),
                    y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
                });
            }
            this.interval = 1;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Logarithmic Base

The Logarithmic Base can be customized using the [logBase](#) property. The default value of this property is 10.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService, LogarithmicService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ StepLineSeriesService, LogarithmicService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" [value]='value'
valueType='Logarithmic' [logBase]='log'>
        <e-rangenavigator-series-collection>

```

```

        <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public value?: Object[];
        public chartData?: Object[];
        public log?: number;
        ngOnInit(): void {
            this.value = [4, 6];
            this.chartData = [];
            for (let i = 0; i < 100; i++) {
                this.chartData.push({
                    x: Math.pow(10, i * 0.1),
                    y: Math.floor(Math.random() * (80 - 30 + 1)) + 30
                });
            }
            this.log = 2;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Date-time

The Range Selector supports the DateTime scale and displays the DateTime values as labels in the specified format.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value'>
        <e-rangenavigator-series-collection>

```

```

        <e-rangenavigator-series [dataSource]='chartData' type='Area'
        xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public periodsValue?: Object[];
        public chartData?: Object[];
        public value?: Object[];
        ngOnInit(): void {
            this.chartData = datasrc;
            this.value=[new Date("2017-08-13"), new Date("2017-12-28")];
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Note: To use date-time scale, inject the `DateTimeService` into the `@NgModule.providers`.

Interval Customization

The `DateTime` intervals can be customized using the `interval` and the `intervalType` properties of the Range Selector. For example, if the `interval` is set to 2 and the `intervalType` is set to years, the interval will be considered to be 2 years.

`DateTime` supports the following interval types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours
- Minutes

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
    imports: [

```

```

        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' intervalType='Months' [interval]='interval'>
        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
  })
  export class AppComponent implements OnInit {
    public chartData?: Object[];
    public value?: Object[];
    public interval?: number;
    ngOnInit(): void {
      this.chartData = datasrc;
      this.value=[new Date("2017-08-13"), new Date("2017-12-28")];
      this.interval = 2;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label Format

The [labelFormat](#) property is used to format and parse the date to all globalize format.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' labelFormat='y/M/d'>
        <e-rangenavigator-series-collection>

```



```

        <e-rangenavigator-series [dataSource]='chartData' type='Area'
        xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public chartData?: Object[];
        public value?: Object[];
        ngOnInit(): void {
            this.chartData = datasrc;
            this.value=[new Date("2017-08-13"), new Date("2017-12-28")];
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The following table shows the results of applying some common DateTime 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 the day format.
new Date(2000, 03, 10)	yMd	04/10/2000	The date is displayed in the 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	The 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.

Period selector in Angular Range navigator component

The period selector allows to select a range with specified periods.

Periods

An array of objects that allows the users to specify pre-defined time intervals. The [interval](#) property specifies the count value of the button, and the [text](#) property specifies the text to be displayed on the button. The [intervaltype](#) property allows the users to customize the interval type, and it supports the following types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours
- Minutes
- Seconds

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, PeriodSelectorService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { chartData } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, PeriodSelectorService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[periodSelectorSettings]='periodsValue'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='close' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public periodsValue?: Object[];
  public chartData?: Object[];
  public tooltip?: Object[];
  public labelFormat?: string;
  ngOnInit(): void {
    this.periodsValue = {
      periods: [
        { text: '1M', interval: 1, intervalType: 'Months' }, { text:
'3M', interval: 3, intervalType: 'Months' },
        { text: '6M', interval: 6, intervalType: 'Months' }, { text:
'YTD' },
        { text: '1Y', interval: 1, intervalType: 'Years' },
        { text: '2Y', interval: 2, intervalType: 'Years', selected:
true },
        { text: 'All' }
      ]
    } as any;
    this.chartData = chartData;
  }
}
```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

To use the period selector feature, inject the `PeriodSelectorService` into the `@NgModule.providers`.

Positioning period selector

The `position` property allows the users to position the period selector at the **Top** or **Bottom**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, PeriodSelectorService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { chartData } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, PeriodSelectorService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[periodSelectorSettings]='periodsValue'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='close' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public periodsValue?: Object[];
  public chartData?: Object[];
  ngOnInit(): void {
    this.periodsValue = {
      position: 'Top',
      periods: [
        { text: '1M', interval: 1, intervalType: 'Months' }, { text:
'3M', interval: 3, intervalType: 'Months' },
        { text: '6M', interval: 6, intervalType: 'Months' }, { text:
'YTD' },
        { text: '1Y', interval: 1, intervalType: 'Years' },
        { text: '2Y', interval: 2, intervalType: 'Years', selected:
true },
        { text: 'All' }
      ]
    }
  }
}
```

```

    ]
    } as any;
    this.chartData = chartData;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Height

The [height](#) property allows the users to specify the height of the period selector. The default value of the height property is **43px**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, PeriodSelectorService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { chartData } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, PeriodSelectorService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[periodSelectorSettings]='periodsValue'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='close' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public periodsValue?: Object[];
  public chartData?: Object[];
  ngOnInit(): void {
    this.periodsValue = {
      height: 65,
      periods: [
        { text: '1M', interval: 1, intervalType: 'Months' }, { text:
'3M', interval: 3, intervalType: 'Months' },
        { text: '6M', interval: 6, intervalType: 'Months' }, { text:
'YTD' },
        { text: '1Y', interval: 1, intervalType: 'Years' },

```

```

        { text: '2Y', interval: 2, intervalType: 'Years', selected:
true
        },
        { text: 'All' }
    ]
} as any;
this.chartData = chartData;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Visibility of range navigator

The [disableRangeSelector](#) property allows the users to display only the period selector and not the Range Selector.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, PeriodSelectorService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { chartData } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, PeriodSelectorService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[disableRangeSelector]='visibility' [periodSelectorSettings]='periodsValue'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='close' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public periodsValue?: Object[];
  public chartData?: Object[];
  public visibility?: boolean;
  ngOnInit(): void {
    this.periodsValue = {
      periods: [

```

```

        { text: '1M', interval: 1, intervalType: 'Months' }, { text:
'3M', interval: 3, intervalType: 'Months' },
        { text: '6M', interval: 6, intervalType: 'Months' }, { text:
'YTD' },
        { text: '1Y', interval: 1, intervalType: 'Years' },
        { text: '2Y', interval: 2, intervalType: 'Years', selected:
true },
        { text: 'All' }
    ]
  } as any;
  this.chartData = chartData;
  this.visibility = true;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [LightWeight](#)

Labels in Angular Range navigator component

Multilevel labels

The multi-level labels for the Range Selector can be enabled by setting the [enableGrouping](#) property to **true**. This is restricted to the DateTime axis alone.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-
charts'
import { AreaSeriesService, DateTimeService, RangeTooltipService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" labelPosition='Outside'
valueType='DateTime' [value]='value' intervalType='Quarter'
[enableGrouping]='grouping' [dataSource]='chartData' xName='x' yName='y'
tooltip='tooltip'></ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public data: object[] = [];

```

```

public value: number = 0;
public point: object = {};
public chartData?: Object[];
public values?: Object[];
public grouping?: boolean;
public tooltip?: Object;
ngOnInit(): void {
    this.chartData = [];
    for (let j = 1; j < 1090; j++) {
        this.value += (Math.random() * 10 - 5);
        this.value = this.value < 0 ? Math.abs(this.value) : this.value;
        this.point = { x: new Date(2000, 0, j), y: this.value, z:
(this.value + 10) };
        this.chartData.push(this.point);
    };
    this.values = [new Date("2001, 1,1"), new Date("2002,1,1")];
    this.grouping = true;
    this.tooltip= { enable: true };
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Grouping

The multi-level labels can be grouped using the [groupBy](#) property with the following interval types:

- Auto
- Years
- Quarter
- Months
- Weeks
- Days
- Hours
- Minutes
- Seconds

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, RangeTooltipService } from
'@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
imports: [
ChartModule, RangeNavigatorModule

```

```

    ],
    providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" labelPosition='Outside'
    valueType='DateTime' [value]='value' intervalType='Quarter'
    [enableGrouping]='grouping' groupBy='Years' [dataSource]='chartData'
    xName='x' yName='y' tooltip='tooltip'></ejs-rangenavigator>`
  })
  export class AppComponent implements OnInit {
    public data: object[] = [];
    public value?: number = 0;
    public point: object = {};
    public chartData?: Object[];
    public values?: Object[];
    public grouping?: boolean;
    public tooltip?: Object;
    ngOnInit(): void {
      this.chartData = [];
      for (let j = 1; j < 1090; j++) {
        (this.value as number) += (Math.random() * 10 - 5) as number;
        this.value = (this.value as number) < 0 ? Math.abs(this.value as
number) : this.value;
        this.point = { x: new Date(2000, 0, j), y: this.value, z:
((this.value as number) + 10) };
        this.chartData.push(this.point);
      };
      this.values = [new Date("2001, 1,1"), new Date("2002,1,1")];
      this.grouping = true;
      this.tooltip= { enable: true };
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Smart labels

The [labelIntersectAction](#) property is used to avoid overlapping of labels. The following code sample shows the setting of [labelIntersectAction](#) property to **Hide**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from
'syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({

```



```

imports: [
    ChartModule, RangeNavigatorModule
],
providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
standalone: true,
selector: 'app-container',
template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' labelIntersectAction='Hide'
labelFormat='y/M/d'>
    <e-rangenavigator-series-collection>
        <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
        </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
))
export class AppComponent implements OnInit {
    public chartData?: Object[];
    public value?: Object[];
    ngOnInit(): void {
        this.chartData = datasrc;
        this.value=[new Date("2017-08-13"), new Date("2017-12-28")];
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label positioning

By default, the labels can be placed outside the Range Selector. It can also be placed inside the Range Selector using the [labelPosition](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' labelPosition='Inside'>

```

```

        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
    })
    export class AppComponent implements OnInit {
        public chartData?: Object[];
        public value?: Object[];
        ngOnInit(): void {
            this.chartData = datasrc;
            this.value=[new Date("2017-08-13"), new Date("2017-12-28")];
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Labels customization

The font size, color, family, etc. can be customized using the [labelStyle](#) setting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { GetDateTimeData } from './datasource'
@Component({
    imports: [
        ChartModule, RangeNavigatorModule
    ],
    providers: [ AreaSeriesService, DateTimeService ],
    standalone: true,
    selector: 'app-container',
    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' intervalType='Months' [labelFormat]='labelFormat'
[labelStyle]='labelStyle' [dataSource]='chartData' xName='x' yName='y'></ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
    public value?: Object[];
    public chartData?: Object[];
    public tooltip?: Object;
    public labelFormat?: string;
    public labelStyle?: Object;
    ngOnInit(): void {
        this.value = [new Date(2018, 5, 1), new Date(2018, 6, 1)];
    }
}

```

```

    this.chartData = GetDateTimeData(new Date(2018, 0, 1), new Date(2019,
0, 1));
    this.labelFormat = 'MMM';
    this.labelStyle= { color: 'red', size:'10px'};
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Grid tick in Angular Range navigator component

Grid line customization

The gridlines indicate axis divisions by drawing the chart plot. Gridlines include helpful cues to the user, particularly for large or complicated charts. The `width`, `color`, and `dashArray` of the major gridlines can be customized by using the [majorGridLines](#) setting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" [value]='value'
[majorGridLines]='majorGridLines'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='xData' yName='yData' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public majorGridLines?: Object[];
  ngOnInit(): void {
    this.value = [25, 40];
    this.chartData = [
      { xData: 10, yData: 35 }, { xData: 20, yData: 28 },
      { xData: 30, yData: 34 }, { xData: 40, yData: 32 }
    ],
  },
}

```

```

        { xData: 50, yData: 40 }
    ];
    this.majorGridLines = [{ width: 4, color: 'blue', dashArray: '5,5'
}];
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tick line customization

Ticklines are the small lines which is drawn on the axis line representing the axis labels. Ticklines will be drawn outside the axis by default. The **width**, **color**, and **dashArray** of the major ticklines can be customized by using the [majorTickLines](#) setting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" [value]='value'
[majorTickLines]='majorTick'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData'
type='StepLine' xName='xData' yName='yData' width=2>
        </e-rangenavigator-series>
      </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public majorTick?: Object[];
  ngOnInit(): void {
    this.value = [25, 40];
    this.chartData = [
      { xData: 10, yData: 35 }, { xData: 20, yData: 28
    },
      { xData: 30, yData: 34 }, { xData: 40, yData: 32
    },
      { xData: 50, yData: 40 }
    ]
  }
}

```

```

    ];
    this.majorTick = [{ width: 3, color: 'Red' }];
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customization in Angular Range navigator component

Navigator appearance

The Range Selector can be customized by using the [navigatorStyleSettings](#). The [selectedRegionColor](#) property is used to specify the color for the selected region, whereas the [unselectedRegionColor](#) property is used to specify the color for the unselected region.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts';
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts';
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [navigatorStyleSettings]='navigatorStyle' labelFormat='MMM-yy'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public navigatorStyle?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.navigatorStyle = { unselectedRegionColor: 'skyblue',
selectedRegionColor: 'pink' };
  }
}

```

```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Thumb

The thumb property allows to customize the border, fill color, size, and type of thumb. Thumbs can be of two shapes: **Circle** and **Rectangle**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [navigatorStyleSettings]='navigatorStyle' labelFormat='MMM-YY'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public navigatorStyle?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasc;
    this.navigatorStyle = { thumb:{ type: 'Rectangle', border: { width:
2, color: 'red'}, fill: 'pink' }};
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';

```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Border customization

Using the [navigatorBorder](#), the [width](#) and [color](#) of the Range Selector border can be customized.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [navigatorBorder]='navigatorBorder' labelFormat='MMM-yy'>
  <e-rangenavigator-series-collection>
    <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
    </e-rangenavigator-series>
  </e-rangenavigator-series-collection>
</ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public navigatorBorder?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.navigatorBorder = { width: 4, color: 'green'};
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Deferred update

If the [enableDeferredUpdate](#) property is set to **true**, then the changed event will be triggered after dragging the slider. If the [enableDeferredUpdate](#) is **false**, then the changed event will be triggered when dragging the slider. By default, the [enableDeferredUpdate](#) is set to **false**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService,
RangeTooltipService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { datasrc } from './datasource';
import { IChangedEventArgs } from '@syncfusion/ej2-charts';
import { Chart } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService,
RangeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [tooltip]='tooltip'
[enableDeferredUpdate]='enableDeferredUpdate' (changed)='changed($event)'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>
  <ejs-chart #chart style='display:block;' id='chart' height='350'
[primaryXAxis]='primaryXAxis' [tooltip]='tooltip'>
    <e-series-collection>
      <e-series [dataSource]='chartData' type='Area' xName='x'
yName='y' width=2>
      </e-series>
    </e-series-collection>`
})
export class AppComponent implements OnInit {
  @ViewChild('chart')
  public chart?: Chart;
  public periodsValue?: Object[];
  public chartData?: Object[];
  public value?: Object[];
  public tooltip?: Object;
  public enableDeferredUpdate?: boolean;
  public changed?: IChangedEventArgs | any;
  public primaryXAxis?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.primaryXAxis = {
      valueType: 'DateTime',
      edgeLabelPlacement: 'Shift'
    }
  }
}
```



```

    };
    this.legendSettings= { visible: false };
    this.chartData = datasrc;
    this.value=[new Date('2017-09-01'), new Date('2018-02-01')];
    this.tooltip={ enable: true };
    this.enableDeferredUpdate = true;
    this.changed = function (args: any) {
        (this.chart as Chart).primaryXAxis.zoomFactor = args.zoomFactor;
        (this.chart as Chart).primaryXAxis.zoomPosition =
args.zoomPosition;
        (this.chart as Chart).dataBind();
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Allow snapping

The [allowSnapping](#) property toggles the placement of the slider exactly to the left or on the nearest interval.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' labelFormat='MMM-yy' [allowSnapping]='allowSnapping'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];

```

```

public allowSnapping?: boolean;
ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.allowSnapping= true;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Animation

The speed of the animation can be controlled using the [animationDuration](#) property. The default value of the [animationDuration](#) property is **500** milliseconds.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' labelFormat='MMM-yy' [animationDuration]='animationDuration'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public animationDuration?: number;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.animationDuration=2000;
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

See Also

- [Grid and Tick Lines](#)
- [Labels](#)

Tool tip in Angular Range navigator component

The tooltip for sliders are supported by the Range Selector. Sliders are used in the Range Selector to select data from a specific range. The tooltip displays the selected start and end values.

Customization

Tooltip can be customized using the following properties:

- enable - Customizes the visibility of the tooltip.
- fill - Customizes the background color of the tooltip.
- opacity - Customizes the opacity of the tooltip.
- textStyle - Customizes the font size, color, family, style, weight, alignment, and overflow of the tooltip.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, RangeTooltipService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './data';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [tooltip]='tooltip'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
```

```
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
    this.tooltip = { enable: true, displayMode: 'Always', fill: 'red',
opacity: 0.6, textStyle: { style: 'Italic', color: 'blue', size: '12px' } };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Label Format

The `labelFormat` property in the tooltip is used to format and parse the date to all globalize formats.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, RangeTooltipService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './data';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, RangeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
[value]='value' [tooltip]='tooltip'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
      </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  public value?: Object[];
  public chartData?: Object[];
  public tooltip?: Object;
  ngOnInit(): void {
    this.value = [new Date('2017-09-01'), new Date('2018-02-01')];
    this.chartData = datasrc;
```

```

    this.tooltip = { enable: true, displayMode: 'Always', labelFormat:
'y/M/d' };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The following table shows the results of applying some common date and time formats to the `labelFormat` property.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format Property Value	Result	Description
<code>new Date(2000, 03, 10)</code>	<code>EEEE</code>	Monday	The Date is displayed in day format
<code>new Date(2000, 03, 10)</code>	<code>yMd</code>	04/10/2000	The Date is displayed in month/date/year format
<code>new Date(2000, 03, 10)</code>	<code>MMM</code>	Apr	The Shorthand month for the date is displayed
<code>new Date(2000, 03, 10)</code>	<code>hm</code>	12:00 AM	Time of the date value is displayed as label
<code>new Date(2000, 03, 10)</code>	<code>hms</code>	12:00:00 AM	The Label is displayed in hours:minutes:seconds format

R t l in Angular Range navigator component

The Range Selector supports right-to-left (RTL), which can be enabled with the [enableRtl](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit } from '@angular/core';
import { datasrc } from './datasource'
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService ],
  standalone: true,
  selector: 'app-container',

```

```

    template: `<ejs-rangenavigator id="rn-container" valueType='DateTime'
    [value]='value' [enableRtl]='enableRtl' labelFormat='MMM-yy'>
        <e-rangenavigator-series-collection>
            <e-rangenavigator-series [dataSource]='chartData' type='Area'
            xName='x' yName='y' width=2>
            </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
    </ejs-rangenavigator>`
  })
  export class AppComponent implements OnInit {
    public enableRtl?: boolean;
    public chartData?: Object[];
    public value?: Object[];
    ngOnInit(): void {
      this.chartData = datasrc;
      this.value=[new Date('2017-09-01'), new Date('2018-02-01')];
      this.enableRtl = true;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Export print in Angular Range navigator component

Export

The rendered Range Selector can be exported to **JPEG, PNG, SVG, or PDF** format by using the [export](#) method in the Range Selector. This method contains the following parameters:

- **Type** - To specify the export type. The component can be exported to **JPEG, PNG, SVG, or PDF** format.
- **File name** - To specify the file name to export.
- **Orientation** - To specify the orientation type. This is applicable only for PDF export type.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from '@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { bitCoinData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { RangeNavigatorComponent } from '@syncfusion/ej2-angular-charts';
@Component({
  imports: [
    ChartModule, RangeNavigatorModule
  ],

```

```

providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
standalone: true,
  selector: 'app-container',
  template: `<button ej-button id='print'
(click)='export()'>Export</button>
  <ejs-rangenavigator #range id="range" valueType='DateTime'
[value]='value'>
    <e-rangenavigator-series-collection>
      <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
    </e-rangenavigator-series>
    </e-rangenavigator-series-collection>
  </ejs-rangenavigator>`
})
export class AppComponent implements OnInit {
  @ViewChild('range')
  public RangeObj?: RangeNavigatorComponent;
  public periodsValue?: Object[];
  public chartData?: Object[];
  public value?: Object[];
  ngOnInit(): void {
    this.chartData = bitCoinData;
    this.value=[new Date(2017, 8, 1), new Date(2018, 1, 1)];
  }
  export() {
    (this.RangeObj as RangeNavigatorComponent ).export('PNG', 'result',
null as any, [this.RangeObj as RangeNavigatorComponent ]);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Print

The rendered Range Selector can be printed directly from the browser by calling the public method [print](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule, RangeNavigatorModule } from '@syncfusion/ej2-angular-charts'
import { AreaSeriesService, DateTimeService, StepLineSeriesService } from
 '@syncfusion/ej2-angular-charts'
import { Component, OnInit, ViewChild } from '@angular/core';
import { bitCoinData } from './datasource';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { RangeNavigatorComponent } from '@syncfusion/ej2-angular-charts';
@Component({
imports: [

```

```

    ChartModule, RangeNavigatorModule
  ],
  providers: [ AreaSeriesService, DateTimeService, StepLineSeriesService ],
  standalone: true,
  selector: 'app-container',
  template: `<button ej-button id='print' (click)='print()'>Print</button>
    <ejs-rangenavigator #range id="range" valueType='DateTime'
    [value]='value'>
      <e-rangenavigator-series-collection>
        <e-rangenavigator-series [dataSource]='chartData' type='Area'
xName='x' yName='y' width=2>
          </e-rangenavigator-series>
        </e-rangenavigator-series-collection>
      </ejs-rangenavigator>`
  })
export class AppComponent implements OnInit {
  @ViewChild('range')
  public RangeObj?: RangeNavigatorComponent;
  public periodsValue?: Object[];
  public chartData?: Object[];
  public value?: Object[];
  ngOnInit(): void {
    this.chartData = bitCoinData;
    this.value=[new Date(2017, 8, 1), new Date(2018, 1, 1)];
  }
  print() {
    (this.RangeObj as RangeNavigatorComponent ).print();
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Accessibility in Angular Range navigator component

The Range navigator 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 Range navigator 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 Range navigator component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Range navigator component:

- region (role)
- aria-label (attribute)

Keyboard interaction

The Range navigator 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 Range navigator component.

| **Press** | **To do this** |

| --- | --- |

| **Tab** | Moves the focus to the Range navigator element. |

| **Ctrl + P** | Prints the Range navigator. |

Ensuring accessibility

The Range navigator component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Range navigator component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Range navigator component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Ej1 api migration in Angular Range navigator component

This article describes the API migration process of Chart component from Essential JS 1 to Essential JS 2.

RangeNavigator

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD038 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Allow snapping | **Property:** *allowSnapping*

<ej:RangeNavigator
[allowSnapping]="true">

</ej:RangeNavigator> | **Property:** *allowSnapping*

<ejs-rangenavigator

[allowSnapping]='allowSnapping'>
</ejs-rangenavigator>
this.allowSnapping= true; |

| Animation duration | Not Applicable | **Property:** *animationDuration*

<ejs-rangenavigator

[animationDuration]='animationDuration'>
</ejs-

rangenavigator>
this.animationDuration= 2000;|

| Border for range navigator | **Property:** *border*

<ej:RangeNavigator
border.color="green" [border.opacity]="0.5"

[border.width]="2">
</ej:RangeNavigator> | **Property:** *navigatorBorder*

<ejs-

rangenavigator [navigatorBorder]='navigatorBorder'>
</ejs-

rangenavigator>
this.navigatorBorder = { width: 4, color: 'green'}|

| dataSource for range navigator | **Property:** *dataSource*

this.rangeData = function

GetData({});
<ej:RangeNavigator
[dataSource]="rangeData">

</ej:RangeNavigator> | **Property:** *dataSource*

<e-rangenavigator-series

dataSource={data}>
</e-rangenavigator-series>|

| enabling deferred update for range navigator | **Property:** *enableDeferredUpdate*

<ej:RangeNavigator
[enableDeferredUpdate]="true">

</ej:RangeNavigator> | **Property:** *enableDeferredUpdate*

<ejs-rangenavigator

[enableDeferredUpdate]='enableDeferredUpdate'>
</ejs-

rangenavigator>
this.enableDeferredUpdate = true |

| multilevel level labels | **Property:** *labelSettings.higherLevelLabels*

this.labelSettings= {

higherLevel: { } };
<ej:RangeNavigator
[labelSettings]="labelSettings">

```

<br/></ej:RangeNavigator> | Property: enableGrouping <br/><br/><ejs-rangenavigator
[enableGrouping]='grouping'><br/></ejs-rangenavigator><br/>this.grouping = true;|

|enabling scroll bar| Property: enableScrollBar
<br/><br/><ej:RangeNavigator<br/>[enableScrollBar]= "true"> <br/></ej:RangeNavigator> | Not
Applicable|

|enabling auto resizing| Property: enableAutoResizing
<br/><br/><ej:RangeNavigator<br/>enableAutoResizing="true"> <br/></ej:RangeNavigator> | Not
Applicable|

|enabling isResponsive| Property: isResponsive
<br/><br/><ej:RangeNavigator<br/>isResponsive="true"> <br/></ej:RangeNavigator> | Not
Applicable|

|enabling RTL for range navigator| Property: enableRtl
<br/><br/><ej:RangeNavigator<br/>enableRtl="true"> <br/></ej:RangeNavigator> | Property:
enableRtl <br/><br/><ejs-rangenavigator [enableRtl]='enableRtl'><br/></ejs-
rangenavigator><br/>this.enableRtl = true;|

|interval for range navigator| Property: valueAxisSettings.interval <br/><br/>this.valueAxisSettings=
{Interval: 5 };<br/><ej:RangeNavigator<br/>[valueAxisSettings]="this.valueAxisSettings">
<br/></ej:RangeNavigator> | Property: interval <br/><br/><ejs-rangenavigator Interval=5
}'><br/></ejs-rangenavigator>|

|intervaltype for range navigator| Property: valueAxisSettings.intervalType
<br/><br/>this.valueAxisSettings= {intervalType:
'Years'};<br/><ej:RangeNavigator<br/>[valueAxisSettings]="this.valueAxisSettings">
<br/></ej:RangeNavigator> | Property: intervalType <br/><br/><ejs-rangenavigator
intervalType='Years'><br/></ejs-rangenavigator>|

|labelformat for range navigator| Not applicable| Property: labelFormat <br/><br/><ejs-
rangenavigator [labelFormat]='labelFormat'><br/></ejs-rangenavigator><br/>this.labelFormat =
'MMM';|

|label intersect action for range navigator| Not applicable| Property: labelIntersectAction
<br/><br/><ejs-rangenavigator labelIntersectAction='Hide'><br/></ejs-rangenavigator>|

|labelStyle range navigator| Property: valueAxisSettings.font <br/><br/>this.valueAxisSettings= {font: { }
};<br/><ej:RangeNavigator<br/>[valueAxisSettings]="this.valueAxisSettings">
<br/></ej:RangeNavigator> | Property: labelStyle <br/><br/><ejs-rangenavigator
[labelStyle]='labelStyle'><br/></ejs-rangenavigator><br/>this.labelStyle= { color: 'red', size:'10px'};|

|locale of range navigator| Property: locale <br/><br/><ej:RangeNavigator<br/>locale="fr-FR">
<br/></ej:RangeNavigator> | Property: locale <br/><br/><ejs-rangenavigator locale='en-
US'><br/></ejs-rangenavigator>|

|major grid lines of range navigator| Property: valueAxisSettings.majorGridLines
<br/><br/>this.valueAxisSettings= { majorGridLines: { width: 2, color: 'red' }
};<br/><ej:RangeNavigator<br/>[valueAxisSettings]="this.valueAxisSettings">
<br/></ej:RangeNavigator> | Property: majorGridLines <br/><br/><ejs-rangenavigator

```

[majorGridLines]='majorGridLines'></ej:range-navigator>
this.majorGridLines={ width: 4, color: 'blue', dashArray: '5,5' };|

|margin of range navigator| Not Applicable| **Property:** *margin*

<ej:range-navigator [margin]='margin'></ej:range-navigator>
this.margin={ };|

|maximum value of range navigator| **Property:** *valueAxisSettings.range.max*

this.valueAxisSettings= { range: { max: 2 } };
<ej:RangeNavigator
[valueAxisSettings]="this.valueAxisSettings">
</ej:RangeNavigator>| **Property:** *maximum*

<ej:range-navigator maximum={34}></ej:range-navigator>|

|minimum value of range navigator| **Property:** *valueAxisSettings.range.min*

this.valueAxisSettings= { range: { min: 2 } };
<ej:RangeNavigator
[valueAxisSettings]="this.valueAxisSettings">
</ej:RangeNavigator>| **Property:** *minimum*

<ej:range-navigator minimum={4}></ej:range-navigator>|

|query for data source of range navigator| Not Applicable| **Property:** *query*

<ej-range-navigator query=" "></ej-range-navigator>|

|Secondary label alignment of range navigator| Not Applicable| **Property:** *secondaryLabelAlignment*

<ej-range-navigator secondaryLabelAlignment='Far'></ej-range-navigator>|

|Skeleton of range navigator axis| Not Applicable| **Property:** *skeleton*

<ej-range-navigator skeleton=" "></ej-range-navigator>|

|skeleton type of range navigator axis| Not Applicable| **Property:** *skeletonType*

<ej-range-navigator skeletonType=" "></ej-range-navigator>|

|Theme of range navigator| **Property:** *theme*

<ej:RangeNavigator
theme=" ">
</ej:RangeNavigator>| **Property:** *theme*

<ej-range-navigator theme=" "></ej-range-navigator>|

|Default selector value range navigator| **Property:** *selectedRangeSettings*

this.selectedRangeSettings= {start:"", end:""};
<ej:RangeNavigator
[selectedRangeSettings]="selectedRangeSettings">
</ej:RangeNavigator>| **Property:** *value*

<ej-range-navigator value=[2, 10]></ej-range-navigator>|

|Value type of range navigator| **Property:** *valueType*

<ej:RangeNavigator
valueType='DateTime'>
</ej:RangeNavigator>| **Property:** *valueType*

<ej-range-navigator valueType='Logarithmic'></ej-range-navigator>|

|Width of range navigator| **Property:** *size.width*

<ej:RangeNavigator
size={ width: '200'}>
</ej:RangeNavigator>| **Property:** *width*

<ej-range-navigator width='400'></ej-range-navigator>|

|Height of range navigator| **Property:** *size.height*

<ej:RangeNavigator
size={ height: '200'}>
</ej:RangeNavigator>| **Property:** *height*

<ej-range-navigator height='400'></ej-range-navigator>|

| Series settings for range navigator | **Property:** *seriesSettings*

this.seriesSettings= {
};
<ej:RangeNavigator
seriesSettings={seriesSettings}>
</ej:RangeNavigator> | Not
Applicable |

| Range settings for range navigator | **Property:** *rangeSettings*

this.rangeSettings= { start: 3,
end: 6 };
<ej:RangeNavigator
rangeSettings={rangeSettings}>

</ej:RangeNavigator> | Not Applicable |

| Scroll range settings for range navigator | **Property:** *scrollRangeSettings*

this.scrollRangeSettings= { start: 3, end: 6
};
<ej:RangeNavigator
scrollRangeSettings={scrollRangeSettings}>

</ej:RangeNavigator> | Not Applicable |

Series

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| animation | **Property:** *enableAnimation*

<ej:RangeNavigator
enableAnimation="true">

</ej:RangeNavigator> | **Property:** *animation.enable*

<ejs-rangenavigator
[animation]='animation'>
</ejs-rangenavigator>
this.animation={ enable: true} |

| Border for range navigator series | **Property:** *border*

this.series = [{ border: { color:
'transparent', width: 2 } }];
<ej:RangeNavigator
series={series}>

</ej:RangeNavigator> | **Property:** *border*

<e-rangenavigator-series
border={
color: 'pink', width: 2}>
</e-rangenavigator-series> |

| dataSource for range navigator | **Property:** *series.dataSource*

this.series = [{ dataSource: [{]
}];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator> | **Property:**
series.dataSource

<e-rangenavigator-series
dataSource={data}>
</e-
rangenavigator-series> |

| query for data source of range navigator | Not Applicable | **Property:** *query*

<e-
rangenavigator-series
dataSource={data} query="">
</e-rangenavigator-series> |

| series type for range navigator | **Property:** *series.type*

this.series = [{ type: "
}];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator> | **Property:** *series.type*

<e-rangenavigator-series
type="">
</e-rangenavigator-series> |

| series xName for range navigator | **Property:** *series.xName*

this.series = [{ xName: "
}];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator> | **Property:**
series.xName

<e-rangenavigator-series
xName="">
</e-rangenavigator-
series> |

| series yName for range navigator | **Property:** *series.yName*

this.series = [{ yName: "
}];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator> | **Property:**
series.yName

<e-rangenavigator-series
yName="">
</e-rangenavigator-
series> |

|series fill color for range navigator| **Property:** *series.fill*

this.series = [{ fill: " " }];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator>| **Property:** *series.fill*

<e-rangenavigator-series
fill="">
</e-rangenavigator-series>|

|series width for range navigator| **Property:** *series.width*

this.series = [{ width: '2' }];
<ej:RangeNavigator
series={series}>
</ej:RangeNavigator>| **Property:** *series.width*

<e-rangenavigator-series
width="">
</e-rangenavigator-series>|

|series dashArray for range navigator| Not Applicable| **Property:** *series.dashArray*

<e-rangenavigator-series
dashArray='10,5'>
</e-rangenavigator-series>|

StyleSettings

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

|Style settings of range navigator| **Property:** *navigatorStyleSettings*

this.navigatorStyleSettings = { leftThumbTemplate: 'left' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator>| **Property:** *navigatorStyleSettings*

<ejs-rangenavigator
[navigatorStyleSettings]='navigatorStyleSettings'>
</ejs-rangenavigator>
this.navigatorStyleSettings = { leftThumbTemplate: 'left' };|

|Selected region color of range navigator| **Property:** *selectedRegionColor*

this.navigatorStyleSettings = { selectedRegionColor: 'red' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator>| **Property:** *selectedRegionColor*

<ejs-rangenavigator
[navigatorStyleSettings]='navigatorStyleSettings'>
</ejs-rangenavigator>
this.navigatorStyleSettings = { selectedRegionColor: 'red' };|

|UnSelected region color of range navigator| **Property:** *unselectedRegionColor*

this.navigatorStyleSettings = { unselectedRegionColor: 'red' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator>| **Property:** *unselectedRegionColor*

<ejs-rangenavigator
[navigatorStyleSettings]='navigatorStyleSettings'>
</ejs-rangenavigator>
this.navigatorStyleSettings = { unselectedRegionColor: 'red' };|

|Thumb color of range navigator| **Property:** *thumbColor*

this.navigatorStyleSettings = { thumbColor: 'red' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator>| **Property:** *thumbSettings*

<ejs-rangenavigator
[navigatorStyleSettings]='navigatorStyleSettings'>
</ejs-rangenavigator>
this.navigatorStyleSettings = { thumbSettings: 'red' };|

|Selected region opacity of range navigator| **Property:** *selectedRegionOpacity*

this.navigatorStyleSettings = { selectedRegionOpacity: 0.4 };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator>| Not Applicable|

| Unselected region opacity of range navigator | **Property:** *UnselectedRegionOpacity*

```
<br/><br>this.navigatorStyleSettings = { UnselectedRegionOpacity: 0.4
};<br/><ej:RangeNavigator<br/>[navigatorStyleSettings]="navigatorStyleSettings">
<br/></ej:RangeNavigator> | Not Applicable |
```

| Background for thumb | **Property:** *background*

this.navigatorStyleSettings = { background: 'red' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

| border for thumb | **Property:** *border*

this.navigatorStyleSettings = { border: { color: 'red', width: 2 } };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

| Highlightsettings for range navigator | **Property:** *highlightSettings*

this.navigatorStyleSettings = { highlightSettings: { } };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

| Selected style settings for range navigator | **Property:** *selectionSettings*

this.navigatorStyleSettings = { selectionSettings: { } };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

| Left thumb template for range navigator | **Property:** *leftThumbTemplate*

this.navigatorStyleSettings = { leftThumbTemplate: '' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

| Right thumb template for range navigator | **Property:** *rightThumbTemplate*

this.navigatorStyleSettings = { rightThumbTemplate: '' };
<ej:RangeNavigator
[navigatorStyleSettings]="navigatorStyleSettings">
</ej:RangeNavigator> | Not Applicable |

Tooltip

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| tooltip | **Property:** *visible*

this.tooltipSettings = { visible: true };
<ej:RangeNavigator
[tooltipSettings]= "Tooltip">
</ej:RangeNavigator> | **Property:** *enable*

<ejs-rangenavigator
[tooltip]='tooltip'
</ejs-rangenavigator>
this.tooltip= { enable: true } |

| background color of tooltip | **Property:** *backgroundColor*

this.tooltipSettings = { backgroundColor: 'red' };
<ej:RangeNavigator
[tooltipSettings]= "Tooltip">
</ej:RangeNavigator> | **Property:** *fill*

<ejs-rangenavigator
[tooltip]='tooltip'
</ejs-rangenavigator>
this.tooltip= { fill: 'pink' } |

| Font style of tooltip | **Property:** *font*

this.tooltipSettings = { font: 'red' };
<ej:RangeNavigator
[tooltipSettings]= "Tooltip">


```

<br/></ej:RangeNavigator> | Property: textStyle <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { textStyle: 'pink' }|
| Label format of tooltip | Property: labelFormat <br/><br/>this.tooltipSettings = {labelFormat:
'yMd'};<br/><ej:RangeNavigator<br/>[tooltipSettings]= "Tooltip">
<br/></ej:RangeNavigator> | Property: format <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { format: 'yMd'}|
| Display mode of tooltip | Property: tooltipDisplayMode <br/><br/>this.tooltipSettings = {
tooltipDisplayMode: 'always'};<br/><ej:RangeNavigator<br/>[tooltipSettings]= "Tooltip">
<br/></ej:RangeNavigator> | Property: displayMode <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { displayMode: 'Always' }|
| Template of tooltip | Not Applicable | Property: template <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { template: '<div>Chart</div>'
}|
| Border of tooltip | Not Applicable | Property: border <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { border: { color: 'red', width: 2}
}|
| Border of tooltip | Not Applicable | Property: opacity <br/><br/><ej:rangenavigator
<br/>[tooltip]='tooltip'<br/></ej:rangenavigator><br/>this.tooltip= { opacity: 0.5 }|

```

Period Selector

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

```

| period Selector position | Not Applicable | Property: periodSelectorSettings.position <br/><br/><ej-
rangnavigator <br/>[periodSelectorSettings]='periodsValue'<br/></ej-
rangnavigator><br/>this.periodsValue = { }|

```

Methods

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

```

| Print | Not Applicable | Property: print() <br/><br/><br/><ej:rangnavigator <br/>#range
id="range"<br/></ej:rangnavigator><br/>print()<br/>this.RangeObj.print();}|

```

```

| Export | Not Applicable | Property: export() <br/><br/><br/><ej:rangnavigator <br/>#range
id="range"<br/></ej:rangnavigator><br/>export()<br/>this.RangeObj.export('PNG', 'result', null,
[this.RangeObj]);}|

```

Events

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Fires before loading the RangeNavigator. | **Property:** *load*

```
<br/><br/><ej:RangeNavigator<br/>(load)="load($event)">
<br/></ej:RangeNavigator><br/>load(sender){ };| Property: load <br/><br/><ejs-rangenavigator
<br/>load={this.load.bind(this)}<br/></ejs-rangenavigator><br/> public load(): void { };|
```

| Fires before loading the RangeNavigator. | **Property:** *loaded*

```
<br/><br/><ej:RangeNavigator<br/>(loaded)="onLoaded($event)">
<br/></ej:RangeNavigator><br/>loaded(sender){ };| Property: loaded <br/><br/><ejs-rangenavigator
<br/>loaded={this.loaded.bind(this)}<br/></ejs-rangenavigator><br/> public loaded(): void { };|
```

| Fires when the value changes in range navigator | **Property:** *rangeChanged*

```
<br/><br/><ej:RangeNavigator<br/>(rangeChanged)="onRangeChanged($event)">
<br/></ej:RangeNavigator><br/>onRangeChanged(sender){ };| Property: changed <br/><br/><ejs-
rangenavigator <br/>changed={this.changed.bind(this)}<br/></ejs-rangenavigator><br/> public
changed(): void { };|
```

| Fires after the RangeNavigator | Not Applicable | **Property:** *resized*

<ejs-rangenavigator

resized={this.resized.bind(this)}
</ejs-rangenavigator>
 public resized(): void { };|

| Fires before tooltip in RangeNavigator | Not Applicable | **Property:** *tooltipRender*

<ejs-
rangenavigator
tooltipRender={this.tooltipRender.bind(this)}
</ejs-
rangenavigator>
 public tooltipRender(): void { };|

| Fires before period render in the RangeNavigator | Not Applicable | **Property:** *selectorRender*

<ejs-rangenavigator
selectorRender={this.selectorRender.bind(this)}
</ejs-
rangenavigator>
 public selectorRender(): void { };|

| Fires when scrollStart the RangeNavigator | **Property:** *scrollStart*

```
<br/><br/><ej:RangeNavigator<br/>(scrollStart)="onScrollStart($event)">
<br/></ej:RangeNavigator><br/>onScrollStart(sender){ };| Not Applicable |
```

| Fires when scrollEnd the RangeNavigator | **Property:** *scrollEnd*

```
<br/><br/><ej:RangeNavigator<br/>(scrollEnd)="onScrollEnd($event)">
<br/></ej:RangeNavigator><br/>onScrollEnd(sender){ };| Not Applicable |
```

| Fires when selected range Start the RangeNavigator | **Property:** *selectedRangeStart*

```
<br/><br/><ej:RangeNavigator<br/>(selectedRangeStart)="onSelectedRangeStart($event)">
<br/></ej:RangeNavigator><br/>onSelectedRangeStart(sender)({ );| Not Applicable |
```

| Fires when selected range ends the RangeNavigator | **Property:** *selectedRangeEnd*

```
<br/><br/><ej:RangeNavigator<br/>(selectedRangeEnd)="onSelectedRangeEnd($event)">
<br/></ej:RangeNavigator><br/>onSelectedRangeEnd(sender)({ );| Not Applicable |
```

| Fires when scroll range changed in the RangeNavigator | **Property:** *scrollChanged*

```
<br/><br/><ej:RangeNavigator<br/>(scrollChanged)="onScrollChanged($event)"><br/></ej:Rang
eNavigator><br/>onScrollChanged({ );| Not Applicable |
```

| Fires when click in the RangeNavigator | **Property:** *click*

```
<br/><br/><ej:RangeNavigator<br/>(click)="onClick($event)"><br/></ej:RangeNavigator><br/>on
Click({ );| Not Applicable |
```

| Fires when right click in the RangeNavigator | **Property:** *rightClick*

```
<br/><br/><ej:RangeNavigator<br/>(rightClick)="onRightClick($event)"><br/></ej:RangeNavigator><br/>onRightClick(){ }; | Not Applicable |
```

| Fires when double click in the RangeNavigator | **Property:** *doubleClick*

```
<br/><br/><ej:RangeNavigator<br/>(doubleClick)="onDoubleClick($event)"><br/></ej:RangeNavigator><br/>onDoubleClick(){ }; | Not Applicable |
```

Range Slider

Getting started with Angular Range Slider component

The Slider component is available in `@syncfusion/ej2-angular-inputs` package. Utilize this package to render the Slider Component.

Setting up angular project

Angular provides the easiest way to set angular CLI projects using [Angular CLI](#) tool.

Install the CLI application globally to your machine.

```
`bash
```

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

```
,
```

Create a new application

```
`bash
```

```
ng new syncfusion-angular-app
```

```
,
```

By default, it install the CSS style base application. To setup with SCSS, pass `--style=SCSS` argument on create project.

Example code snippet.

```
`bash
```

```
ng new syncfusion-angular-app --style=SCSS
```

```
,
```

Navigate to the created project folder.

```
`bash
```

```
cd syncfusion-angular-app
```

```
,
```

Installing Syncfusion RangeSlider Package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)

2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages($\geq 20.2.36$) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-inputs](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs --save
```

```
,
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-inputs@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs@ngcc --save
```

```
,
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
```

```
@syncfusion/ej2-angular-inputs:"20.2.38-ngcc"
```

```
,
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Slider Module

After installing the package, the component modules are available to configure into your application from installed syncfusion package. Syncfusion Angular package provides two different types of `ngModules`.

Refer to [Ng-Module](#) to learn about `ngModules`.

Refer to the following snippet to import the Slider module in `app.module.ts` from the [@syncfusion/ej2-angular-inputs](#).

```
`typescript
```

```
import { BrowserModule } from '@angular/platform-browser';
```

```
import { NgModule } from '@angular/core';
```

```
import { FormsModule } from '@angular/forms';
```

```
import { HttpClientModule } from '@angular/http';
```

```
import { AppComponent } from './app.component';
```

```
// Imported syncfusion Slider module from inputs package
import { SliderModule } from '@syncfusion/ej2-angular-inputs';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpClientModule,
    // Registering EJ2 Slider Module
    SliderModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
`
```

Adding Syncfusion Slider Component

Add the Slider component snippet in `app.component.ts` as follows.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>
    Hello Angular, Syncfusion Angular UI Slider!
    </h1>
    <ejs-slider id='slider' [value]=30></ejs-slider>
    `
})
export class AppComponent {
}
```

Adding Slider CSS reference

Add Slider component styles as given in the `angular-cli.json` file within the apps -> styles section.

Note: If you are using Angular 6 project, add the changes in `angular.json` file.

```
`typescript
{
  "apps": [
    {
      "styles": [
        "styles.css",
        "../node_modules/@syncfusion/ej2-angular-inputs/styles/material.css",
        "../node_modules/@syncfusion/ej2-base/styles/material.css",
        "../node_modules/@syncfusion/ej2-buttons/styles/material.css",
        "../node_modules/@syncfusion/ej2-popups/styles/material.css"
      ]
    }
  ]
}
```

The below example shows a basic `Slider` example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [value]=30></ejs-slider>
      </div>
    </div>`,
  styleUrls: ['./index.css'],
})
export class AppComponent {
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Types

The types of Slider are as follows:

| **Types** | **Usage** |

| --- | --- |

| **Default** | Shows a default Slider to select a single value. |

| **MinRange** | Displays the shadow from the start value to the current selected value. |

| **Range** | Selects a range of values. It also displays the shadow in-between the selection range. |

Both the Default Slider and Min-Range Slider have same behavior that is used to select a single value.

In Min-Range Slider, a shadow is considered from the start value to current handle position. But the Range Slider contains two handles that is used to select a range of values and a shadow is considered in between the two handles.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <div class="sliderwrap">
          <label class="labeltext">Default</label>
          <ejs-slider id='default' [value]=30></ejs-slider>
        </div>
        <div class="sliderwrap">
          <label class="labeltext">MinRange</label>
          <ejs-slider id='minrange' [type]="minType"
[value]='minValue'></ejs-slider>
        </div>
        <div class="sliderwrap">
          <label class="labeltext">Range</label>
          <ejs-slider id='range' [type]="rangeType"
[value]='rangeValue'></ejs-slider>
        </div>
      </div>
    </div>
```

```

    </div>
  </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public minType: string = "MinRange";
  public rangeType: string = "Range";
  public minValue: number = 30;
  public rangeValue: number[] = [30, 70];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customization*Orientation*

The Slider can be displayed, either in horizontal or vertical orientation. By default, the Slider renders in horizontal orientation.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [orientation]= 'orientation'
[value]=30></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public orientation: string ="Vertical";
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip

The Slider displays the tooltip to indicate the current value by clicking the Slider bar or drag the Slider handle. The Tooltip position can be customized by using the **placement** property. Also decides the tooltip display mode on a page, i.e., on hovering, focusing, or clicking on the Slider handle and it always remains/displays on the page.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [type]= 'type' [tooltip] = 'tooltip'
[value]=30></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public type: string ="MinRange";
  public tooltip: Object ={ placement: 'After', isVisible: true, showOn:
'Always' };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Buttons

The Slider value can be changed by using the Increase and Decrease buttons. In Range Slider, by default the first handle value will be changed while clicking the button. Change the handle focus and press the button to change the last focused handle value.

After enabling the slider buttons if the 'Tab' key is pressed, the focus goes to the handle and not to the button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```



```
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [type]= 'type' [showButtons]=true [value]
= 'value' ></ejs-slider>
      </div>
    </div>`,
  styleUrls: ['./index.css']
})
export class AppComponent {
  public type: string = "Range";
  public value: number[] = [30, 70];
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

See Also

[Slider Types](#)

[Slider Formatting](#)

[Orientation Slider](#)

[Ticks in Slider](#)

[Tooltip in Slider](#)

Schematics in Angular Range slider component

Angular schematics is a workflow tool that allows you generate component, modules, and resolve dependency issues. The main goal of the schematics is ease of use and development in angular environment.

EJ2 Slider supports Angular schematic's module injection, component generation, automation dependency installation, styles imports, etc.

Getting started

Note: Angular schematics supports only from the Angular CLI v6. So, check your version by running `ng --version`. If it is below version 6, update your CLI by running the following command: `npm install -g @angular/cli`.

In order to work with Angular schematics, create an Angular CLI application. Run the following command to create a CLI application.

`

```
ng new angular-application
```

`

After running the above command and all the dependency modules installed, we can generate EJ2 Slider component using schematics.

Dependency and Module injection using Schematics

Using schematics, we can perform dependency and module injection of the EJ2 inputs package `@syncfusion/ej2-angular-inputs` automatically. Run the following command in the root of the application.

`

```
ng add @syncfusion/ej2-angular-inputs --modules=slider
```

`

If we ignore the `--modules=slider` flag, it will import all the modules present (ColorPickerModule, UploaderModule, etc.) inside `@syncfusion/ej2-angular-inputs` package.

The above command will do the following,

- Installs the `@syncfusion/ej2-angular-inputs` package, and add an entry in `package.json` file.
- Imports the `SliderModule` module in the `app.module.ts`, and add an entry in the `import` property of the `@NgModule` decorator.

Component generation using Schematics

Angular Schematics can be used to generate component, module file, etc. In the same way, Slider components can also be generated.

By using the Schematics to generate EJ2 Slider, the time for configuring components is significantly reduced and it is made ready for development immediately. To generate EJ2 Slider component with specific features, refer to the following table.

The general syntax for the `ng generate` command is `ng generate @syncfusion/<component-package-name>:<componentName-featureName> --name=<your-desired-name>`

Feature Name	Schematics command
:-: ---	
Default Slider	<code>ng generate @syncfusion/ej2-angular-inputs:slider-default --name=default-slider</code>
MinRange Slider	<code>ng generate @syncfusion/ej2-angular-inputs:slider-minrange --name=minrange-slider</code>
Range Slider	<code>ng generate @syncfusion/ej2-angular-inputs:slider-range --name=range-slider</code>

```
| Ticks Slider | ng generate @syncfusion/ej2-angular-inputs:slider-ticks --name=ticks-slider
|
| Limits Slider | ng generate @syncfusion/ej2-angular-inputs:slider-limits --name=limits-slider
|
| Vertical Slider | ng generate @syncfusion/ej2-angular-inputs:slider-vertical --name=vertical-
slider |
```

The above commands will do the following,

- Generate the Slider with specific features mentioned in the `src/app` with folder name of the `name` property passed.
- Import the generated component into the `app.module.ts` file, and add an entry in the `declarations` array in the `@NgModule` decorator.

Note: It is not required to run the above commands only after the `ng add @syncfusion/ej2-angular-inputs`, but it is required to have `@syncfusion/ej2-angular-inputs` installed.

Ticks in Angular Range slider component

The Ticks in Slider supports you to easily identify the current value/values of the Slider. It contains `smallStep` and `largeStep`. The value of the major ticks alone will be displayed in the slider. In order to enable/disable the small ticks, use the `showSmallTicks` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [tooltip]= 'tooltip' [ticks]='ticks'
[value] = 'value' ></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public value: number =30;
  public tooltip: Object = { placement: 'Before', isVisible: true, showOn:
'Always' };
  public ticks: Object = { placement: 'After', largeStep: 20, smallStep: 10,
showSmallTicks: true };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Step

When the Slider is moved, it increases/decreases the value based on the step value. By default, the value is increased/decreased by 1. Use the `step` property to change the increment step value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [value]='value' [tooltip]="tooltip" [ticks]
= 'ticks' [step] = 'step' ></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public value: number = 30;
  public tooltip: Object = { placement: 'Before', isVisible: true, showOn:
'Always' };
  public ticks: Object = { placement: 'After', largeStep: 20, smallStep: 10,
showSmallTicks: true };
  public step: number = 10;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Min and Max

Enables the minimum/starting and maximum/ending value of the Slider, by using the `min` and `max` property. By default, the minimum value is 1 and maximum value is 100. In the following sample the slider is rendered with the min value as 100 and max value as 1000.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [min]= 'min' [max]= 'max' [value]=
'value' [tooltip]='tooltip'
[ticks] = 'ticks' ></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public min: number = 0;
  public max: number = 100;
  public value: number = 30;
  public ticks: Object = { placement: 'After', largeStep: 20, smallStep:
10, showSmallTicks: true };
  public tooltip: Object = { placement: 'Before', isVisible: true, showOn:
'Always' };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Format in Angular Range slider component

The **format** feature used to customize the units of Slider values to desired format. The formatted values will also be applied to the ARIA attributes of the slider. There are two ways of achieving formatting in slider.

- Use the format API of slider which utilizes our [Internationalization](#) to format values.
- Customize using the events namely **renderingTicks** and **tooltipChange**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
```

```
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [min]=0 [max]=100 [value]=30
[tooltip]="tooltipData" [ticks]="ticksData" ></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public tooltipData: Object = { isVisible: true, format: 'C2' };
  public ticksData: Object = { placement: 'After', format: 'C2', largeStep:
20, smallStep: 10, showSmallTicks: true };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Using format API

In this method, we have different predefined formatting styles like Numeric (N), Percentage (P), Currency (C) and # specifiers. In this below example, we have formatted the ticks and tooltip values into percentage.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [min]=0 [max]=1 [step]=.01 [value]=.3
[tooltip]="tooltipData" [ticks]="ticksData" ></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
```

```

})
export class AppComponent {
  public tooltipData: Object = { placement: 'Before', isVisible: true,
showOn: 'Always', format: 'P0' };
  public ticksData: Object = { placement: 'After', largeStep: .2,
smallStep: .1, showSmallTicks: true, format: 'P0' };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Using Events

In this method, we will be retrieving the values from the slider events then process them to desired formatted the values.

In this sample we have customized the ticks values into weekdays as one formatting and tooltip values into day of the week as another formatting.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
import { SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-
angular-inputs';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
<div id='container'>
  <div class='wrap'>
    <ejs-slider id='slider' #default [min]=0 [max]=6 [value]=2
[tooltip]="tooltipData" [ticks]="ticksData"
(tooltipChange)="tooltipChangeHandler($event)"
(renderingTicks)="renderingTicksHandler($event)" ></ejs-slider>
  </div>
</div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public tooltipData: Object = { placement: 'Before', isVisible: true };
  public ticksData: Object = { placement: 'After', largeStep: 1 };
  tooltipChangeHandler(args: SliderTooltipEventArgs | any): void {
    // Weekdays Array
    let daysArr: string [] =
['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thrusday', 'Friday', 'Saturday'];

```

```
// Customizing each ticks text into weekdays
args.text = daysArr[parseFloat(args.value)];
}
renderingTicksHandler(args: SliderTickEventArgs | any): void {
    // Customizing tooltip to display the Day (in numeric) of the week
    args.text = 'Day ' + (Number(args.value) + 1).toString();
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Limits in Angular Range slider component

The slider limits restrict the slider thumb between a particular range. This is used if higher or lower value affects the process or product where the slider is being used.

The following are the six options in the slider's limits object. Each API in the limits object is optional.

- enabled: Enables the limits in the Slider.
- minStart: Sets the minimum limit for the first handle.
- minEnd: Sets the maximum limit for the first handle.
- maxStart: Sets the minimum limit for the second handle.
- maxEnd: Sets the maximum limit for the second handle.
- startHandleFixed: Locks the first handle.
- endHandleFixed: Locks the second handle.

Default and MinRange Slider limits

There is only one handle in the Default and MinRange Slider, so minStart, minEnd, and startHandleFixed options can be used.

When the limits are enabled in the Slider, the limited area becomes darken. So you can differentiate the allowed and restricted area.

Refer to the following snippet to enable the limits in the Slider.

```
`typescript
```

```
.....
```

```
limits: { enabled: true, minStart: 10, minEnd: 40 }
```

```
.....
```

```
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
```



```

@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [type]='type' [min]='min' [max]='max'
[value]= 'value' [tooltip]='tooltip'
        [limits]='limits'></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public type: string ="MinRange";
  public value: number = 30;
  public tooltip: object= { isVisible: true };
  public min: number = 0;
  public max: number = 100;
  public limits: object = { enabled: true, minStart: 10, minEnd: 40 };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Range Slider limits

In the range slider, both handles can be restricted and locked from the limit's object. In this sample, the first handle is limited between 10 and 40, and the second handle is limited between 60 and 90.

`typescript

.....

```
limits: { enabled: true, minStart: 10, minEnd: 40, maxStart: 60, maxEnd: 90 }
```

.....

`

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [

    SliderModule

```

```

    ],
    standalone: true,
    selector: 'my-app',
    template: `
      <div id='container'>
        <div class='wrap'>
          <ejs-slider id='slider' [type]='type' [min]='min' [max]='max'
[value]= 'value' [tooltip]='tooltip'
          [limits]='limits'></ejs-slider>
        </div>
      </div>`,
    styleUrls:['./index.css']
  })
  export class AppComponent {
    public type: string = "Range";
    public value: number[] = [30, 70];
    public tooltip: object= { isVisible: true };
    public min: number = 0;
    public max: number = 100;
    public limits: object = { enabled: true, minStart: 10, minEnd: 40,
maxStart: 60, maxEnd: 90 };
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Handle lock

The movement of slider handles can be locked by enabling the startHandleFixed and endHandleFixed properties in the limit's object.

In this sample, the movement of both slider handles has been locked.

`typescript

.....

```
limits: { enabled: true, startHandleFixed: true, endHandleFixed: true }
```

.....

`

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    SliderModule
  ],

```

```

standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [type]='type' [min]= 'min' [max]= 'max'
[value]= 'value' [tooltip]='tooltip'
[limits]='limits'></ejs-slider>
      </div>
    </div>`,
  styleUrls: ['./index.css']
})
export class AppComponent {
  public type: string = "Range";
  public value: number[] = [30, 70];
  public tooltip: object = { isVisible: true };
  public min: number = 0;
  public max: number = 100;
  public limits: object = { enabled: true, startHandleFixed: true,
endHandleFixed: true };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Style in Angular Range slider 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 slider track

Use the following CSS to customize the slider track.

```

`css
.e-control-wrapper.e-slider-container.e-horizontal .e-slider-track {
background: #007bff;
height: 3px;
}
`

```

Customizing the slider handle

Use the following CSS to customize the slider handle properties.

```

`css
.e-control-wrapper.e-slider-container .e-slider .e-handle {
background-color: #f9920b;
}
`

```

```
border-radius: 50%;
```

```
border: 0;
```

```
}
```

```
,
```

Customizing the slider limits

Use the following CSS to customize the slider limits.

```
`css
```

```
.e-control-wrapper.e-slider-container.e-horizontal .e-limits {
```

```
background-color: rgba(69, 100, 233, 0.46);
```

```
}
```

```
,
```

Customizing the slider ticks

Use the following CSS to customize the slider ticks.

```
`css
```

```
.e-scale .e-tick.e-custom::before {
```

```
content: '\e967';
```

```
position: absolute;
```

```
}
```

```
,
```

Customizing the slider buttons

Use the following CSS to customize the slider buttons.

```
`css
```

```
.e-control-wrapper.e-slider-container .e-slider-button {
```

```
background: #007bff;
```

```
height: 25px;
```

```
width: 25px;
```

```
}
```

```
,
```

Accessibility in Angular Range Slider component

The Range Slider 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 Range Slider 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 Range Slider component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Range Slider component:

| Attributes | Purpose |

| --- | --- |

| `role=slider` | Used to convey a significant and contextual message to the user. |

| `aria-valuemin` | Indicates the Minimum value of the slider. |

- | `aria-valuemax` | Indicates the Maximum value of the slider. |
- | `aria-valuenow` | Indicates the current value of the slider. |
- | `aria-valuetext` | Returns the current text of the slider. |
- | `aria-orientation` | Indicates whether the Slider is oriented horizontally or vertically. |
- | `aria-label` | Provides an accessible name for the Slider, serving as label text for the Slider's left and right buttons (for increment and decrement). |

Keyboard interaction

The Range Slider 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 Range Slider component.

- | **Press** | **To do this** |
- | --- | --- |
- | **Right Arrow/Up Arrow** | **Increase the Slider value.** |
- | **Left Arrow/Down Arrow** | **Decrease the Slider value.** |
- | **Home** | **Moves to the start value (for Range Slider when the second thumb is focused and the Home key is pressed, it moves to the first thumb value).** |
- | **<kbd>End** | **Moves to the end value (for Range Slider when the first thumb is focused and the End key is pressed, it moves to the second thumb value).** |
- | **Page Up** | **Increases the Slider by `largeStep` value.** |
- | **Page Down** | **Decreases the Slider by `largeStep` value.** |

Ensuring accessibility

The Range Slider component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Range Slider component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Range Slider component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Ej1 api migration in Angular Range slider component

This article describes the API migration process of Slider component from Essential JS 1 to Essential JS 2

- | Behavior | API in Essential JS 1 | API in Essential JS 2 |
- | --- | --- | --- |
- | Max value | **Property:** `maxValue`

 `<ej-slider id='slider' maxValue="100"></ej-slider>` | **Property:** `max`

 `<ejs-slider id='slider' max='100'></ejs-slider>` |

| Min value | **Property:** *minValue*

 <ej-slider id='slider' minValue="100"></ej-slider> |
Property: *min*

 <ejs-slider id='slider' min='20'></ejs-slider> |

| Step | **Property:** *incrementStep, largeStep, smallStep, showSmallTicks*

 <ej-slider
id='slider' incrementStep="20" smallStep="5" largeStep="40" showSmallTicks="true"></ej-
slider>| **Property:** *ticks*

 public ticks = { placement: 'After', largeStep: 20, smallStep:
10, showSmallTicks: true };

 <ejs-slider id='slider' [ticks]='ticks'></ejs-slider> |

| Type | **Property:** *sliderType*

 <ej-slider id='slider' sliderType='range'></ej-slider> |
Property: *type*

 <ejs-slider id='slider' type='Range'></ejs-slider> |

| Tooltip | **Property:** *showTooltip*

 <ej-slider id='slider' showTooltip="true"></ej-slider>
| **Property:** *tooltip*

 public tooltip = { placement: 'Before', isVisible: true, showOn:
'Always' };

 <ejs-slider id='slider' [tooltip]='tooltip'></ejs-slider> |

| RTL | **Property:** *enableRTL*

 <ej-slider id='slider' enableRTL="true"></ej-slider> |
Property: *enableRtl*

 <ejs-slider id='slider' enableRtl='false'></ejs-slider> |

| Custom values | **Not Applicable** | **Property:** *customValues*

 public customValues =
['Mon', 'Tue', 'Wed'];

 <ejs-slider id='slider' customValues='customValues'></ejs-
slider> |

| Limit the slider movement | **Not Applicable** | **Property:** *limits*

 public limits = { enabled:
true, minStart: 10, minEnd: 40 };

 <ejs-slider id='slider' type='MinRange'
limits='limits'></ejs-slider> |

| Disable | **Method:** *disable*

 <ej-slider #slider id='slider'></ej-slider>

this.sliderObj.disable(); | **Property:** *enabled*

 <ejs-slider id='slider'
enable='false'></ejs-slider>
 |

| Enable | **Method:** *enable*

 <ej-slider #slider id='slider'></ej-slider>

this.sliderObj.enable(); | **Property:** *enabled*

 <ejs-slider #slider id='slider'></ejs-slider>

 this.sliderObj.enable = true; |

| Set Value | **Method:** *setValue(value, [enableAnimation])*

 <ej-slider #slider
id='slider'></ej-slider>

 this.sliderObj.setValue(10); | **Property:** *value*

 <ejs-
slider #slider id='slider'></ejs-slider>

 this.sliderObj.value = 50; |

| Get Value | **Method:** *getValue()*

 <ej-slider #slider id='slider'></ej-slider>

this.sliderObj.getValue(); | **Property:** *value*

 <ejs-slider #slider id='slider'></ejs-slider>

 let sliderValue = this.sliderObj.value; |

| Destroy | **Not Applicable** | **Method:** *destroy()*

 <ejs-slider #slider id='slider'
enable='false'></ejs-slider>

 this.sliderObj.destroy(); |

| Change | **Event:** *change*

 public change(args: Event): void { }

 <ej-slider
#slider id='slider' (change)='change(\$event)'></ej-slider> | **Event:** *change*

 public
change(args: Event): void { }

 <ejs-slider #slider id='slider'
(change)='change(\$event)'></ejs-slider> |

```
| Create | Event: create <br/> <br/> public create(args: Event): void { } <br/> <br/> <ej-slider
#slider id='slider' (create)='create($event)'></ej-slider> | Event: created <br/> <br/> public
created(args: Event): void { } <br/> <br/> <ejs-slider #slider id='slider'
(created)='created($event)'></ejs-slider> |
```

```
| Slide | Event: slide <br/> <br/> public slide(args: Event): void { } <br/> <br/> <ej-slider #slider
id='slider' (slide)='slide($event)'></ej-slider> | Event: change <br/> <br/> public change(args:
Event): void { } <br/> <br/> <ejs-slider #slider id='slider' (change)='change($event)'></ejs-slider>
|
```

```
| Start | Event: start <br/> <br/> public start(args: Event): void { } <br/> <br/> <ej-slider #slider
id='slider' (start)='start($event)'></ej-slider> | Event: created <br/> <br/> public created(args:
Event): void { } <br/> <br/> <ejs-slider #slider id='slider' (created)='created($event)'></ejs-slider>
|
```

```
| Stop | Event: stop <br/> <br/> public stop(args: Event): void { } <br/> <br/> <ej-slider #slider
id='slider' (stop)='stop($event)'></ej-slider> | Event: changed <br/> <br/> public changed(args:
Event): void { } <br/> <br/> <ejs-slider #slider id='slider' (changed)='changed($event)'></ejs-
slider> |
```

```
| Rendered Ticks | Not Applicable | Event: renderedTicks <br/> <br/> public renderedTicks(args:
Event): void { } <br/> <br/> <ejs-slider #slider id='slider'
(renderedTicks)='renderedTicks($event)'></ejs-slider> |
```

How To

Customize slider bar in Angular Range slider component

Slider appearance can be customized through CSS. By overriding the slider CSS classes, you can customize the slider bar. The slider bar can be customized with different themes. By default, slider have class name `e-slider-track` for bar. The class can be overridden with our own color values like the following code snippet.

```
`css
.e-control.e-slider .e-slider-track .e-range {
background: linear-gradient(left, #e1451d 0, #fdff47 17%, #86f9fe 50%, #2900f8 65%, #6e00f8 74%,
#e33df9 83%, #e14423 100%);
}
`
```

You can also apply background color for a certain range depending upon slider values, using change event.

```
`typescript
change(args: SliderChangeEventArgs) => {
if (args.value > 0 && args.value <= 25) {
// Change handle and range bar color to green when
(sliderHandle as HTMLElement).style.backgroundColor = 'green';
}
```



```

(sliderTrack as HTMLElement).style.backgroundColor = 'green';
} else if (args.value > 25 && args.value <= 50) {
// Change handle and range bar color to royal blue
(sliderHandle as HTMLElement).style.backgroundColor = 'royalblue';
(sliderTrack as HTMLElement).style.backgroundColor = 'royalblue';
} else if (args.value > 50 && args.value <= 75) {
// Change handle and range bar color to dark orange
(sliderHandle as HTMLElement).style.backgroundColor = 'darkorange';
(sliderTrack as HTMLElement).style.backgroundColor = 'darkorange';
} else if (args.value > 75 && args.value <= 100) {
// Change handle and range bar color to red
(sliderHandle as HTMLElement).style.backgroundColor = 'red';
(sliderTrack as HTMLElement).style.backgroundColor = 'red';
}
}
,

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { SliderComponent, SliderChangeEventArgs } from '@syncfusion/ej2-
angular-inputs';
@Component({
imports: [

    SliderModule
],
standalone: true,
selector: 'my-app',
template: ` <div class="slider_container">
    <div class="slider-labeltext slider_userselect">Height</div>
    <!-- Square slider element -->
    <ejs-slider id='height_slider' [value]='value' [min]='min'
[max]='max' ></ejs-slider>
    </div>
    <div class="slider_container">
    <div class="slider-labeltext slider_userselect">Gradient
color</div>
    <ejs-slider id='gradient_slider' [value]='gradient_value'
[min]='min' [max]='max' [type]='range' ></ejs-slider>
    </div>
    <div class="slider_container">

```

```

        <div class="slider-labeltext slider_userselect">Dynamic thumb and
        selection bar color</div>
        <ejs-slider id='dynamic_color_slider' [value]='dynamic_value'
        [min]='min' [max]='max' [type]='range' (created)='created()'
        (change)='change($event)'></ejs-slider>
    </div>`,
    styleUrls : ["/app.component.css"],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    public value: number = 30;
    public min: number = 0;
    public max: number = 100;
    public gradient_value: number = 30;
    public range: string = 'MinRange';
    public sliderTrack?: HTMLElement;
    public sliderHandle?: HTMLElement;
    public dynamic_value: number = 30;
    // Handler used for slider created event
    created() {
        this.sliderTrack = (document.getElementById('dynamic_color_slider')
as HTMLElement).querySelector('.e-range') as HTMLElement;
        this.sliderHandle = (document.getElementById('dynamic_color_slider')
as HTMLElement).querySelector('.e-handle') as HTMLElement;
        (this.sliderHandle as HTMLElement).style.backgroundColor = 'green';
        (this.sliderTrack as HTMLElement).style.backgroundColor = 'green';
    }
    // Handler used for slider change event
    change(args: SliderChangeEventArgs | any) {
        if (args.value > 0 && args.value <= 25) {
            // Change handle and range bar color to green when
            (this.sliderHandle as HTMLElement).style.backgroundColor =
'green';
            (this.sliderTrack as HTMLElement).style.backgroundColor =
'green';
        } else if (args.value > 25 && args.value <= 50) {
            // Change handle and range bar color to royal blue
            (this.sliderHandle as HTMLElement).style.backgroundColor =
'royalblue';
            (this.sliderTrack as HTMLElement).style.backgroundColor =
'royalblue';
        } else if (args.value > 50 && args.value <= 75) {
            // Change handle and range bar color to dark orange
            (this.sliderHandle as HTMLElement).style.backgroundColor =
'darkorange';
            (this.sliderTrack as HTMLElement).style.backgroundColor =
'darkorange';
        } else if (args.value > 75 && args.value <= 100) {
            // Change handle and range bar color to red
            (this.sliderHandle as HTMLElement).style.backgroundColor = 'red';
            (this.sliderTrack as HTMLElement).style.backgroundColor = 'red';
        }
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize slider limits in Angular Range slider component

Slider appearance can be customized via CSS. By overriding the slider CSS classes, the slider limit bar can be customized. Here, the limit bar is customized with different background color. By default, the slider has class `e-limits` for limits bar. You can override the class with our own color values as given in the following code snippet.

```
`css

.e-slider-container.e-horizontal .e-limits {
background-color: rgba(69, 100, 233, 0.46);
}
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { SliderModule, SliderComponent, LimitDataModel, SliderType,
TicksDataModel, TooltipDataModel } from '@syncfusion/ej2-angular-inputs';
import { SliderTooltipEventArgs, SliderTickEventArgs } from '@syncfusion/ej2-
inputs';
@Component({
imports: [

    SliderModule
],
standalone: true,
selector: 'my-app',
template: ` <div class='sliderwrap'>
    <label class="labeltext">MinRange Slider With Limits</label>
    <ejs-slider id='minrange' #minrange [value]='minValue'
[min]='min' [max]='max' [tooltip]='tooltip' [ticks]='ticks' [type]='minType'
[limits]='minRangeLimits'></ejs-slider>
</div>
    <div class='sliderwrap'>
    <label class="labeltext">Range Slider With Limits</label>
    <ejs-slider id='range' #range [value]='rangeValue' [min]='min'
[max]='max' [tooltip]='tooltip' [ticks]='ticks' [type]='rangetype'
[limits]='rangeLimits'></ejs-slider>
</div>`,
styleUrls : ['./app.component.css']
})
export class AppComponent {
    @ViewChild('minrange')
    public minrangeObj?: SliderComponent;
    @ViewChild('range')
    public rangeObj?: SliderComponent;
```

```

public min: number = 0;
public max: number = 100;
public minValue: number = 30;
public rangeValue: number[] = [30, 70];
public tooltip: TooltipDataModel = {
  placement: 'Before',
  isVisible: true
};
public ticks: TicksDataModel = {
  placement: 'After',
  largeStep: 20,
  smallStep: 5,
  showSmallTicks: true
};
public minType: SliderType = 'MinRange';
public rangetype: SliderType = 'Range';
public minRangeLimits: LimitDataModel = { enabled: true, minStart: 10,
minEnd: 40 };
public rangeLimits: LimitDataModel = { enabled: true, minStart: 10,
minEnd: 40, maxStart: 60, maxEnd: 90 };
// Eventlisteners to lock first handle of the both sliders
public fixOne(args: any): void {
  (this.minrangeObj as SliderComponent).limits.startHandleFixed =
args.checked;
  (this.rangeObj as SliderComponent).limits.startHandleFixed =
args.checked;
}
// Eventlisteners to lock second handle of the both sliders
public fixSecond(args: any): void {
  (this.minrangeObj as SliderComponent).limits.endHandleFixed =
args.checked;
  (this.rangeObj as SliderComponent).limits.endHandleFixed =
args.checked;
}
// Eventlisteners to change limit values for both sliders
public minStartChange(args: any): void {
  (this.minrangeObj as SliderComponent).limits.minStart = args.value;
  (this.rangeObj as SliderComponent).limits.minStart = args.value;
}
public minEndChange(args: any): void {
  (this.minrangeObj as SliderComponent).limits.minEnd = args.value;
  (this.rangeObj as SliderComponent).limits.minEnd = args.value;
}
public maxStartChange(args: any): void {
  (this.minrangeObj as SliderComponent).limits.maxStart = args.value;
  (this.rangeObj as SliderComponent).limits.maxStart = args.value;
}
public maxEndChange(args: any): void {
  (this.minrangeObj as SliderComponent).limits.maxEnd = args.value;
  (this.rangeObj as SliderComponent).limits.maxEnd = args.value;
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize slider thumb in Angular Range slider component

Slider appearance can be customized through CSS. By overriding the slider CSS classes, you can customize the thumb. By default, slider has unique class `e-handle` for slider thumb. You can override the following class as per your requirement. Here, in the sample, the slider thumb has been customized to square, circle, oval shapes, and background image has also been customized.

`typescript

```
.e-control.e-slider .e-handle {
background-image: url('https://ej2.syncfusion.com/demos/src/slider/images/thumb.png');
background-color: transparent;
height: 25px;
width: 25px;
}
```

/ square slider /

```
.square_slider.e-control.e-slider .e-handle {
border-radius: 0%;
background-color: #f9920b;
border: 0;
}
```

/ circle slider /

```
.circle_slider.e-control.e-slider .e-handle {
border-radius: 50%;
background-color: #f9920b;
border: 0;
}
```

/ oval slider /

```
.oval_slider.e-control.e-slider .e-handle {
height: 25px;
width: 8px;
top: 3px;
border-radius: 15px;
background-color: #f9920b;
```

```

}
`

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { SliderComponent } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class="slider_container">
      <div class="labelText slider-userselect">Square</div>
      <ejs-slider id='square_slider' [value]='value' [min]='min'
[max]='max'></ejs-slider>
    </div>
    <div class="slider_container">
      <div class="labelText slider-userselect">Circle</div>
      <ejs-slider id='circle_slider' [value]='value' [min]='min'
[max]='max'></ejs-slider>
    </div>
    <div class="slider_container">
      <div class="labelText slider-userselect">Oval</div>
      <ejs-slider id='oval_slider' [value]='value' [min]='min'
[max]='max'></ejs-slider>
    </div>
    <div class="slider_container">
      <div class="labelText slider-userselect">Custom image</div>
      <ejs-slider id='image_slider' [value]='value' [min]='min'
[max]='max' [ticks]='ticks'></ejs-slider>
    </div>`,
    styleUrls : ['./app.component.css'],
    encapsulation: ViewEncapsulation.None
  })
export class AppComponent {
  public value: number = 30;
  public min: number = 0;
  public max: number = 100;
  public ticks: Object = {
    placement: 'After'
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize slider ticks label in Angular Range slider component

Slider view can be customized via CSS. By overriding the slider CSS classes, you can customize the ticks. The ticks in slider allows you to easily identify the current value/values of the slider. It contains [smallStep](#) and [largeStep](#). By default, slider has class `e-tick` for slider ticks. You can override the class as per your requirement.

Refer to the following code snippet to render ticks.

```
`typescript
.e-scale .e-tick.e-custom::before {
content: '\e967';
position: absolute;
}
`
```

Here, the color for rendered ticks has been applied through `nth-child(child_number)`. The color is applied to the value of the `child_number` in the slider.

```
`typescript
.e-scale :nth-child(1)::before {
color: red;
}
`
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { SliderComponent } from '@syncfusion/ej2-angular-inputs';
import { SliderTickRenderedEventArgs, SliderTickEventArgs, Placement } from '@syncfusion/ej2-inputs';
@Component({
imports: [
SliderModule
],
standalone: true,
selector: 'my-app',
template: `<div class="slider_container" id="slider_wrapper">
<div class="slider_labelText userselect">Dynamic ticks
color</div>
<ejs-slider id='ticks_slider' [value]='value' [min]='min'
[max]='max' [type]='type' [step]='step' [ticks]='ticks'
(renderingTicks)=renderingTicks($event) ` ></ejs-slider>
</div>
<div class="slider_container">
```

```

        <div class="slider_labelText userselect">Ticks with legends</div>
        <ejs-slider id='slider' [value]='value' [min]='min' [max]='max'
[type]='type' [ticks]='slider_ticks'
(renderedTicks)='renderedTicks($event)'></ejs-slider>
    </div>`,
    styleUrls : ["/app.component.css"],
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    public count: number = 1;
    public value: number = 30;
    public min: number = 0;
    public max: number = 100;
    public step: number = 5;
    public type: string = 'MinRange';
    public ticks: Object = { placement: 'Before', largeStep: 20 };
    public slider_ticks: Object = { placement: 'Both', largeStep: 20,
smallStep: 5 };
    public renderingTicks(args: SliderTickEventArgs) {
      if (args.tickElement.classList.contains('e-large')) {
        args.tickElement.classList.add('e-custom');
      }
    }
    public renderedTicks(args: SliderTickRenderedEventArgs) {
      let li: any = args.ticksWrapper.getElementsByClassName('e-large');
      let remarks: any = ['Very Poor', 'Poor', 'Average', 'Good', 'Very
Good', 'Excellent'];
      for (let i: number = 0; i < li.length; ++i) {
        (li[i].querySelectorAll('.e-tick-both')[1] as
HTMLElement).innerText = remarks[i];
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Date range slider in Angular Range slider component

The Date formatting can be achieved in **ticks** and **tooltip** using **renderingTicks** and **tooltipChange** events respectively.

The process of formatting is explained in the below sample.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
import { SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-
angular-inputs';
@Component({

```



```

imports: [
    SliderModule
],
standalone: true,
selector: 'my-app',
template: `
<div id='container'>
  <div class='wrap'>
    <ejs-slider id='slider' [min]="min" [max]="max" [value]="value"
step=86400000 [tooltip]="tooltipData" [ticks]="ticksData" [showButtons]=true
(tooltipChange)="tooltipChangeHandler($event)"
(renderingTicks)="renderingTicksHandler($event)"></ejs-slider>
  </div>
</div>`,
styleUrls:['./index.css']
})
export class AppComponent {
  public tooltipData: Object = { placement: 'Before', isVisible: true };
  public ticksData: Object = { placement: 'After', largeStep: 2 * 86400000 };

  public min: number = new Date("2013-06-13").getTime();
  public max: number = new Date("2013-06-21").getTime();
  public step: number = 86400000;
  public value: number = new Date("2013-06-15").getTime();
  tooltipChangeHandler(args: SliderTooltipEventArgs): void {
    let totalMiliSeconds = Number(args.text);
    // Converting the current milliseconds to the respective date in
    desired format
    let custom: any = { year: "numeric", month: "short", day: "numeric" };
    args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
    custom);
  }
  renderingTicksHandler(args: SliderTickEventArgs): void {
    let totalMiliSeconds = Number(args.value);
    // Converting the current milliseconds to the respective date in
    desired format
    let custom: any = { year: "numeric", month: "short", day: "numeric" };
    args.text = new Date(totalMiliSeconds).toLocaleDateString("en-us",
    custom);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Form slider with formvalidator in Angular Range slider component

We can validate the Slider component using our **FormValidator**. The following steps walk-through for slider validation.

- Render Slider component inside form.
- Bind [changed](#) event in the Slider component to validate the slider value when the value changes.
- Initialize and render FormValidator for the form using form ID.
- Set required property in the FormValidator `rules` collection.

Here, we set the [min](#) property of Slider which sets the minimum value in Slider component and it has hidden input since enable `validateHidden` property as true.

```
`typescript
//slider element
<ejs-slider id='default'></ejs-slider>

// sets required property in the FormValidator rules collection
export class AppComponent {
  @ViewChild('formId') element:any;
  public formObject: FormValidator;
  ngAfterViewInit() {
    let options: FormValidatorModel = {
      rules: {
        'default': {
          validateHidden: true,
          min: [6, "You must select value greater than 5"]
        }
      }
    };
    this.formObject = new FormValidator(this.element.nativeElement, options);
  }
}
```

Form validation done by either ID or name value of Slider component. In above used ID of the slider for validate it.

Using Slider name: Render Slider with name attribute. In the below code snippet we have used name attribute value of 'slider' for form validation.

```
`typescript
//slider element with name attribute
<ejs-slider id='default' name='slider'></ejs-slider>

// sets required property in the FormValidator rules collection
```

```
export class AppComponent {
  @ViewChild('formId') element:any;
  public formObject: FormValidator;
  ngAfterViewInit() {
    let options: FormValidatorModel = {
      rules: {
        'slider': {
          validateHidden: true,
          min: [30, "You must select value greater than 30"]
        }
      }
    };
    this.formObject = new FormValidator(this.element.nativeElement, options);
  }
},
```

- Validate the form using `validate` method and it validates the Slider value with the defined rules collection and returns the result.

If user selects the value less than the minimum value, form will not submit.

```
`typescript
this.formObject.validate();
,
```

- Slider validation can be done during value changes in slider. Refer to the below code snippet.

```
`typescript
public onChanged(): void {
  this.formObject.validate();
}
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
```

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { SliderComponent } from '@syncfusion/ej2-angular-inputs';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
@Component({
  imports: [

    SliderModule,
    FormsModule,
    ReactiveFormsModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
<div id='container'>
  <div class="content-wrapper" style="margin-bottom: 25px">
    <div class="form-title">
      <span>Slider Form Validation</span>
    </div>
    <form #formId class="form-horizontal">
      <div class="form-group">
        <div class="e-float-input">
          <ejs-slider id='default' #default name='slider' [value]=20
(changed)='onChanged()' '
            [ticks]='ticks'></ejs-slider>
        </div>
      </div>
    </form>
    <button ejs-button isPrimary="true" type="submit" id="submit_btn"
(click)="btnClick($event)">Submit</button>
  </div>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('formId') element: any;
  @ViewChild('default') sliderObj?: SliderComponent;
  public formObject?: FormValidator;
  public ticks: Object = {
    placement: 'Before',
    largeStep: 20,
    smallStep: 5,
    showSmallTicks: true
  };
  ngAfterViewInit() {
    let options: FormValidatorModel = {
      rules: {
        'slider': {
          validateHidden: true,
          min: [30, "You must select value greater than 30"]
        }
      }
    };
    this.formObject = new FormValidator(this.element.nativeElement,
options);
  }
}

```

```

public btnClick(args: any): void {
  if ((this.sliderObj as any).value < 5) {
    alert("Please select value greater than 30");
  } else {
    alert("Submitted");
    this.element.nativeElement.submit();
  }
}
public onChanged(): void {
  this.formObject?.validate();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Numeric range slider in Angular Range slider component

The numeric values can be formatted into different decimal digits or fixed number of whole numbers or to represent the units.

The Numeric processing is demonstrated below.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
import { SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-
angular-inputs';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <div class='label'>Slider formatted with unit
representation</div>
        <ejs-slider id='slider' [min]=0 [max]=100 [value]=30
[tooltip]="tooltipData01" [ticks]="ticksData01"></ejs-slider>
      </div>
      <div class='wrap'>
        <div class='label'>Slider formatted with three decimal
specifiers</div>
        <ejs-slider id='slider1' [min]=0.1 [max]=0.2 [step]=0.01
[value]=0.13 [tooltip]="tooltipData02" [ticks]="ticksData02"></ejs-slider>
      </div>
    </div class='wrap'>

```

```

        <div class='label'>Slider formatted with two leading zeros</div>
        <ejs-slider id='slider2' [min]=0 [max]=100 [value]=30
[tooltip]="tooltipData03" [ticks]="ticksData03"></ejs-slider>
    </div>
</div>`,
    styleUrls:['./index.css']
})
export class AppComponent {
    public tooltipData01: Object = { isVisible: true, format: '##.## Km' };
    public ticksData01: Object = { placement: 'After', format: '##.## Km',
largeStep: 20, smallStep: 10, showSmallTicks: true };
    public tooltipData02: Object = { isVisible: true, format: '##.## Km' };
    public ticksData02: Object = { placement: 'After', format: '##.## Km',
largeStep: 0.02, smallStep: 0.01, showSmallTicks: true };
    public tooltipData03: Object = { isVisible: true, format: '00##' };
    public ticksData03: Object = { placement: 'After', format: '00##',
largeStep: 20, smallStep: 10, showSmallTicks: true };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Slider in angular reactive form in Angular Range slider component

Slider validation can be achieved in Angular using [Reactive](#) forms. Here the sample shown slider validation state based on Angular form [classes](#).

Follow below steps to validate slider within reactive forms.

- Create simple Angular reactive form. And add simple [slider](#) component within form.
- Create [form group](#) with slider.

`typescript

```

sliderForm: FormGroup;

constructor(fb: FormBuilder) {
    this.value = 30;
    this.sliderForm = fb.group({
        'slider': [0, Validators.min(10)]
    });
}

```

- Show the validation message, based on validation classes which is added to slider. Refer below code snippet.

```

| Class if true | Class if false | state |
| --- | --- | --- |
| ng-touched | ng-untouched | The control has been visited. |
| ng-dirty | ng-pristine | The control's value has changed. |
| ng-valid | ng-invalid | The control's value is valid. |
,

<div *ngIf="sliderForm.invalid">
slider has <b><i>invalid </i> </b> value and choose value greater than 10.
</div><br/>
<div *ngIf="sliderForm.valid">
Slider has <b><i>valid </i> </b> value {{value}}.
</div><br/>
<div *ngIf="sliderForm.pristine">
Slider having state <b><i>pristine.</i></b> Slider value not yet changed by user. <br/>
</div>
<div *ngIf="sliderForm.dirty">
Slider having state <b><i>dirty.</i> </b> Slider value changed by user.
</div>
<br/>
<button [disabled]="sliderForm.invalid" type="submit">submit</button><br/><br/>
<div class="formresult" [hidden]="!validated">
Slider value choosen as: {{value}}
</div>
,

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { Component, Inject } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { SliderComponent } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [
    SliderModule,
    FormsModule, ReactiveFormsModule
  ],

```

```

standalone: true,
selector: 'my-app',
template: `
<div class="container">
  <form [formGroup]="sliderForm" (ngSubmit)="onSubmit()">
    <div class="form-group">
      <label for="slider">Slider</label>
      <ejs-slider formControlName="slider" id='sliderelement'
[ (ngModel) ]="value" type="MinRange" required></ejs-slider>

      <div *ngIf="sliderForm.invalid">
        slider has <b><i>invalid </i></b> value and choose value greater
than 10.
      </div>
      <div *ngIf="sliderForm.valid">
        Slider has <b><i>valid </i></b> value {{value}}.
      </div><br/>
      <div *ngIf="sliderForm.pristine">
        Slider having state <b><i>pristine.</i></b> Slider value not yet
changed by user.
      </div>
      <div *ngIf="sliderForm.dirty">
        Slider having state <b><i>dirty.</i></b> Slider value changed by
user.
      </div><br/>
      <div *ngIf="sliderForm.untouched">
        Slider having state <b><i>untouched.</i></b> Slider has not
visited by user.
      </div>
      <div *ngIf="sliderForm.touched">
        Slider having state <b><i>touched.</i></b> Slider has been
visited by user.
      </div>
      <br/>
      <button [disabled]="sliderForm.invalid"
type="submit">submit</button><br/><br/>
      <div class="formresult" [hidden]="!validated">
        Slider value choosen as: {{value}}
      </div>
    </div>
  </form>
</div>`,
styleUrls:['./index.css']
})
export class AppComponent {
  value;
  slidervalue?: SliderComponent;
  validated = false;
  sliderForm: FormGroup;
  constructor(@Inject(FormBuilder) public fb: FormBuilder) {
    this.value = 30;
    this.sliderForm = this.fb.group({
      'slider': [0, Validators.min(10)]
    });
  }
  onSubmit() {
    if (this.sliderForm.valid) {

```



```

        this.validated = true;
        console.log('form submitted');
        console.log(this.sliderForm.value.slider);
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Slider validation using template driven forms in Angular Range slider component

Slider can be validated in Angular using [Template-driven](#) forms.

- The following [CSS classes](#) will be added on Slider component based on the action done by user.

| **Class if true** | **Class if false** | **state** |

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

| **ng-touched** | **ng-untouched** | The control has been visited. |

| **ng-dirty** | **ng-pristine** | The control's value has changed. |

| **ng-valid** | **ng-invalid** | The control's value is valid. |

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { SliderModule } from '@syncfusion/ej2-angular-inputs';
import { FormsModule } from '@angular/forms';
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { SliderModule } from '@syncfusion/ej2-angular-inputs';
import { NgForm } from '@angular/forms';
@Component({
  imports: [

    SliderModule,
    FormsModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class="container">
      <form #sliderForm="ngForm" (ngSubmit)="onSubmit()">
        <div class="form-group">
          <label for="slider">Slider</label>
          <ejs-slider name = "slider" id='default' type="MinRange" slider-
            validate [(ngModel)] ='value' required></ejs-slider>

          <div *ngIf="sliderForm.invalid">

```

```

        slider has <b><i>invalid </i> </b> value and choose value greater
        than 10.
    </div>
    <div *ngIf="sliderForm.valid">
        Slider has <b><i>valid </i> </b> value {{value}}.
    </div>
    <br />
    <div *ngIf="sliderForm.pristine">
        Slider having state <b><i>pristine.</i></b> Slider value not yet
        changed by user.
    </div>
    <div *ngIf="sliderForm.dirty">
        Slider having state <b><i>dirty.</i> </b> Slider value changed by
        user.
    </div>
    <br />
    <div *ngIf="sliderForm.untouched">
        Slider having state <b><i>untouched.</i></b> Slider has not
        visited by user.
    </div>
    <div *ngIf="sliderForm.touched">
        Slider having state <b><i>touched.</i> </b> Slider has been
        visited by user.
    </div>
    <br />
    <button [disabled]="sliderForm.invalid"
type="submit">submit</button><br/><br/>
    <div class="formresult" [hidden]="!validated">
        Slider value choosen as: {{value}}
    </div>
</div>

</form>
</div>`,
styleUrls:['./index.css'],
encapsulation: ViewEncapsulation.None
}))
export class AppComponent {
    value?: number;
    validated?: boolean;
    @ViewChild('sliderForm') form?: NgForm;
    onSubmit() {
        this.validated = true;
        console.log(this.form?.valid)
    }
    ngOnInit() {
        this.value = 70;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Slider with ngmodel in Angular Range slider component

Slider component supports one and two-way property binding. Slider two way binding can be achieved through [ngModel](#) Angular directive.

Follow the below steps to perform two-way binding with ngModel.

- Create simple [slider](#) component and binds the value property using ngModel. Refer to the below code snippet.

```
<ejs-slider class="form-control" id='slider' [ticks]="ticks" [(ngModel)]="slidervalue" name="slider"
required #slider="ngModel"></ejs-slider>
```

- Create numeric text box and bind the value using ngModel.

```
<input type="number" id="name" name="name" class="form-control" required
[(ngModel)]="slidervalue" #slider="ngModel">
```

- And name the same variable name in both slider and numeric text box. Which will help to view the two-way binding i.e. changing value in slider will change the numeric textbox value and vice versa.
- Initialize the value of the variable in component file, while will be bound to slider and text box initially. The values will be changed synchronously while changing any one (slider or text-box) value.

`typescript

```
export class AppComponent {
  slidervalue = "30";
  public ticks: Object = {
    placement: 'Before',
    largeStep: 20,
    smallStep: 5,
    showSmallTicks: true
  };
}
```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
import { FormGroup } from '@angular/forms';
@Component({
  imports: [
    SliderModule,
    FormsModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
<div class="container">
  <form #sliderForm="ngForm">
    <div class="form-group">
      <label for="slider">Slider</label>
      <br/> <br/> <br/>
      <ejs-slider class="form-control" id='slider' [ticks]="ticks"
[ (ngModel)]="slidervalue" name="slider" required #slider="ngModel"></ejs-
slider>
      <br/> <br/> <br/>
      <input type="number" id="name" name="name" class="form-control"
required [ (ngModel)]="slidervalue" #slider="ngModel">
    </div>
  </form>
</div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  slidervalue = "30";
  public ticks: Object = {
    placement: 'Before',
    largeStep: 20,
    smallStep: 5,
    showSmallTicks: true
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Time range slider in Angular Range slider component

The time formatting can be achieved same as the date formatting using `renderingTicks` and `change` events. The process of time formatting is explained in the below sample.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
import { SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <ejs-slider id='slider' [min]="min" [max]="max" [value]="value"
[tooltip]="tooltipData" [ticks]="ticksData" [showButtons]=true [step]="step"
        (tooltipChange)='tooltipChangeHandler($event)'
(renderingTicks)='renderingTicksHandler($event)'></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public tooltipData: Object = { placement: 'Before', isVisible: true };
  public ticksData: Object = { placement: 'After', largeStep: 2 * 3600000
};

  public min: number =new Date(2013, 6, 13, 11).getTime();
  public max: number = new Date(2013, 6, 13, 17).getTime();
  public step: number = 3600000;
  public value: number = new Date(2013, 6, 13, 13).getTime();
  tooltipChangeHandler(args: SliderTooltipEventArgs): void {
    let totalMiliSeconds = Number(args.text);
    let custom: any = { hour: '2-digit', minute: '2-digit' };
    args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
  }
  renderingTicksHandler(args: SliderTickEventArgs): void {
    let totalMiliSeconds = Number(args.value);
    let custom: any = { hour: '2-digit', minute: '2-digit' };
    args.text = new Date(totalMiliSeconds).toLocaleTimeString("en-us",
custom);
  }
}

```

MAIN.TS

```

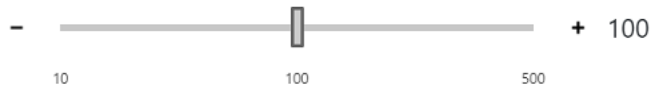
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize slider msword in Angular Range slider component

Slider view can be customized via CSS. By overriding the slider CSS classes, you can customize the slider buttons and thumbs. The ticks in slider allows you to easily identify the value in the slider.

Here we are going to achieve the Slider as like the below screenshot.



By default, slider has even split values in ticks. And we can achieve slider ticks value as like above image. To achieve this we can use [customValues](#) to render the Ticks. Refer to the below code snippet to display needed ticks in slider.

```
`typescript
public renderedTicks(event) {
  event.tickElements.forEach((element) => {
    if (element.title !== '10' && element.title !== '100' && element.title !== '500') {
      element.classList.add('e-display-none');
    }
  })
}
```

To customize the Slider handle, refer to the below code snippet.

```
`css
.slider.e-control.e-slider .e-handle {
  height: 25px;
  width: 8px;
  top: 3px;
  margin-left: -4px;
  border-radius: 1px;
  background-color:#ccc;
}
```

To customize the Slider Button, refer to the below code snippet.

```
`css
.e-control-wrapper.e-slider-container .e-slider-button{
  border: 0px;
}
.e-control-wrapper.e-slider-container.e-horizontal .e-second-button .e-button-icon {
```

background-image:

```
url('data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAYAAABXAvmHAAAAAXNSR0I
Ars4c6QAAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAADKSURBVGhD7dQxDYMw
EIXhc9aicec1KFiaziN4C7MJjOCWaUicXFwihJGeTnqfhMAFxQ+ywAh3flhhL92b9VjAtm3inLu0LcuiZ/XjHUBj
ABoD0BiAxA0BqAxAM18QFuR1ff5HqUUmedZR+dijBJCONE93vvvvgXUHYl/5Uw5wAaA9AYgMYANPMB
7Unc+7lv33dJKeno3DRNMgyDju4Zx/F3UAOesK5rvRCXtpyzntWPcwCNAWgMQGMAGgPQGIDGADTzAfxb
BUvkDXe7zQJva1o6AAAAEIFTkSuQmCC');
```

```
.e-control-wrapper.e-slider-container.e-horizontal .e-first-button .e-button-icon {
```

background-image:

```
url('data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAYAAABXAvmHAAAAAXNSR0I
Ars4c6QAAAAARnQU1BAACxjwv8YQUAAAAJcEhZcwAADsMAAA7DAcdvqGQAAAB1SURBVGhD7dfBCCAgE
AVRk2aszmqsmpMWIltDD/MuyzsbQ4Leu1XC3Z/M5YBNANoBtAMoBIAM4BmAM0A2vmRzTlrkWKMUf
ME9N5rkWktVdMboBIAM4BmAC0+4D9PiVTeAM0AmgE0A2gG0AygGUAzgBYe0NoDs18dSCJjSCYAAAAAS
UVORK5CYII=');
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule,
    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div class="container">
      <div id="text">
        <label for="slider">Slider</label>
        <br/> <br/> <br/>
        <div class="middle">
          <div class="first">
            <ejs-slider id='slider' width="300"
[customValues]='customvalues' [showButtons]=true [(ngModel)]="slidervalue"
name="slider" required #slider="ngModel" [ticks]='ticks'
(change)="change($event)" (renderedTicks)="renderedTicks($event)">
              </ejs-slider>
            </div>
            <div class="second">
              <span for="slider">{{value}}</span>
            </div>
          </div>
        </div>
      </div>`,
  styleUrls: ['./app.component.css']
})
```

```

    })
    export class AppComponent {
        public slidervalue = 3;
        public customvalues = [10, 20, 30, 100, 250, 400, 500];
        public ticks = {
            placement: 'After',
        };
        public value = this.customvalues[this.slidervalue];
        public change(args: any) {
            this.value = this.customvalues[args.value];
        }
        public renderedTicks(event: any) {
            event.tickElements.forEach((element: any) => {
                if (element.title !== '10' && element.title !== '100' && element.title
                !== '500') {
                    element.classList.add('e-display-none');
                }
            })
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show slider from hidden state in Angular Range slider component

This section demonstrates how-to render the Slider component in hidden state and make it visible in button click. We can initialize Slider in hidden state by setting the display as none.

In the sample, by clicking on the button, we can make the Slider visible from hidden state, and we must also call the [refresh](#) method of the Slider to render it properly based on its original dimensions.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewEncapsulation } from '@angular/core';
import { SliderModule, SliderTickEventArgs, SliderTooltipEventArgs } from
 '@syncfusion/ej2-angular-inputs';
/**
 * Default sample
 */
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <button ej2-button id="element">Button</button>

```



```

    <div id='container'>
      <div id="wrap">
        <ejs-slider id='slider' [min]="min" [max]="max" [value]="value"
[tooltip]="tooltipData" [ticks]="ticksData" [showButtons]=true [step]="step"
(tooltipChange)='tooltipChangeHandler($event)'
(renderingTicks)='renderingTicksHandler($event)'></ejs-slider>
      </div>
    </div>,
    styleUrls:['./index.css']
  })
export class AppComponent {
  public tooltipData: Object = { placement: 'Before', isVisible: true };
  public ticksData: Object = { placement: 'After', largeStep: 2 * 3600000
};
  public min: number = new Date(2013, 6, 13, 11).getTime();
  public max: number = new Date(2013, 6, 13, 17).getTime();
  public step: number = 3600000;
  public value: number = new Date(2013, 6, 13, 13).getTime();
  tooltipChangeHandler(args: SliderTooltipEventArgs): void {
    let totalMilliseconds = Number(args.text);
    let custom: any = { hour: '2-digit', minute: '2-digit' };
    args.text = new Date(totalMilliseconds).toLocaleTimeString("en-us",
custom);
  }
  renderingTicksHandler(args: SliderTickEventArgs): void {
    let totalMilliseconds = Number(args.value);
    let custom: any = { hour: '2-digit', minute: '2-digit' };
    args.text = new Date(totalMilliseconds).toLocaleTimeString("en-us",
custom);
  }
  ngOnInit() {
    (document.getElementById('element') as HTMLElement).onclick = () => {
      let slider: HTMLElement = document.getElementById("wrap") as
HTMLElement;
      let ticks: HTMLElement | any = document.getElementById("slider") as
HTMLElement;
      slider.style.display = "block";
      ticks.ej2_instances[0].refresh();
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Reversible Range Slider in Angular

You can create a Range Slider rendered with values in reverse order by setting the [min](#) property to the maximum value and the [max](#) property to the minimum value. An example of how to achieve a reversible Range Slider is shown below

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SliderModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
import { SliderTickEventArgs, SliderTooltipEventArgs } from '@syncfusion/ej2-
angular-inputs';
@Component({
  imports: [

    SliderModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id='container'>
      <div class='wrap'>
        <div class='label'>Slider formatted with unit
representation</div>
        <ejs-slider id='slider' [min]=0 [max]=100 [value]=30
[tooltip]="tooltipData01" [ticks]="ticksData01"></ejs-slider>
      </div>
      <div class='wrap'>
        <div class='label'>Slider formatted with three decimal
specifiers</div>
        <ejs-slider id='slider1' [min]=0.1 [max]=0.2 [step]=0.01
[value]=0.13 [tooltip]="tooltipData02" [ticks]="ticksData02"></ejs-slider>
      </div>
      <div class='wrap'>
        <div class='label'>Slider formatted with two leading zeros</div>
        <ejs-slider id='slider2' [min]=0 [max]=100 [value]=30
[tooltip]="tooltipData03" [ticks]="ticksData03"></ejs-slider>
      </div>
    </div>`,
  styleUrls:['./index.css']
})
export class AppComponent {
  public tooltipData01: Object = { isVisible: true, format: '##.## Km' };
  public ticksData01: Object = { placement: 'After', format: '##.## Km',
largeStep: 20, smallStep: 10, showSmallTicks: true };
  public tooltipData02: Object = { isVisible: true, format: '##.##00' };
  public ticksData02: Object = { placement: 'After', format: '##.##00',
largeStep: 0.02, smallStep: 0.01, showSmallTicks: true };
  public tooltipData03: Object = { isVisible: true, format: '00##' };
  public ticksData03: Object = { placement: 'After', format: '00##',
largeStep: 20, smallStep: 10, showSmallTicks: true };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Reversible order can be achieved with [Horizontal](#) orientation Range Slider by setting [enableRtlLink to the Video](#) as true.

Rating

Getting started with Angular Rating component

This section explains how to create a default Rating and demonstrate the basic usage of the Rating module.

To get started quickly with Angular rating component, you can check out this video:

Dependencies

The list of dependencies required to use the Rating module in your application is given below:

```
`javascript
|-- @syncfusion/ej2-angular-inputs
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-popups
`,`
```

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

```
`
npm install -g @angular/cli
`,`
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
`
ng new my-app
cd my-app
`,`
```

Installing Syncfusion Rating package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(>=20.2.36) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-inputs](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-inputs --save
`
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the [ngcc](#) package use the below.

Add [@syncfusion/ej2-angular-inputs@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-inputs@ngcc --save
`
```

To mention the ngcc package in the [package.json](#) file, add the suffix [-ngcc](#) with the package version as below.

```
`bash
@syncfusion/ej2-angular-inputs:"20.4.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Rating module

Import Rating module into Angular application(app.module.ts) from the package

[@syncfusion/ej2-angular-inputs](#).

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Importing Rating module from Syncfusion ej2-angular-inputs package.
import { RatingModule } from '@syncfusion/ej2-angular-inputs';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, RatingModule ], // Declaration of RatingModule into NgModule.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
```

```

})
export class AppModule { }
`

```

Adding Syncfusion Rating component

Modify the template in `app.component.ts` file to render the Rating component.

```

`typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `<!-- To render Rating. -->
<input ej-rating id="rating" />`
})
export class AppComponent { }
`

```

Adding CSS reference

Add Rating component's styles as given below in `style.css`.

```

`css
@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';
`

```

Running the application

Run the application in the browser using the following command:

```

`
ng serve
`

```

The following example shows a default Rating component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

```

```
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating"/>
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Value

You can set the rating value by using the [value](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3.0"/>
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Precision Modes in Angular Rating Component

You can use the [precision](#) property of the rating component to provide ratings with varying levels of precision.

The precision types of Rating are as follows:

- Full: The rating is increased in whole number increments. For example, if the current rating is 2, the next possible ratings are 3, 4, and so on.
- Half: The rating is increased in increments of 0.5 (half). For example, if the current rating is 2.5, the next possible ratings are 3, 3.5, 4, and so on.
- Quarter: The rating is increased in increments of 0.25 (quarter). For example, if the current rating is 3.75, the next possible ratings are 4, 4.25, 4.5, and so on.
- Exact: The rating is increased in increments of 0.1. For example, if the current rating is 3.9, the next possible ratings are 4, 4.1, 4.2, and so on.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <Label>Full Precision</Label><br />
      <input ej2-rating id="rating1" precision="Full" value="3"
    /><br />
      <Label>Half Precision</Label><br />
      <input ej2-rating id="rating2" precision="Half"
    value="2.5" /><br />
      <Label>Quarter Precision</Label><br />
      <input ej2-rating id="rating3" precision="Quarter"
    value="3.75" /><br />
      <Label>Exact Precision</Label><br />
      <input ej2-rating id="rating4" precision="Exact"
    value="2.3" /><br />
    </div>`
  })
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Labels in Angular Rating Component

You can use the [showLabel](#) property to display a label that shows the current value of the rating. When the [showLabel](#) property is set to `true`, a label will be displayed.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" showLabel='true' value="3"
    />
    </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Label position

The rating component allows you to place the label on the top, bottom, left, or right side of the rating using the [labelPosition](#) property.

The following label positions are supported:

- Top: The label is placed on the top of the rating.
- Bottom: The label is placed on the bottom of the rating.
- Left: The label is placed on the left side of the rating.
- Right: The label is placed on the right side of the rating.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
```



```

imports: [
    FormsModule, RatingModule
],
standalone: true,
selector: 'app-root',
template: `<!-- To Render Rating component. -->
    <div class="wrap">
        <Label>Left Label Position</Label><br/>
        <input ej-s-rating id="rating1" value="3" showLabel="true"
labelPosition="Left" /><br/>
        <Label>Right Label Position</Label><br />
        <input ej-s-rating id="rating2" value="3" showLabel="true"
/><br/>
        <Label>Top Label Position </Label><br />
        <input ej-s-rating id="rating3" value="3" showLabel="true"
labelPosition="Top" /><br/>
        <Label>Bottom Label Position</Label><br />
        <input ej-s-rating id="rating4" value="3" showLabel="true"
labelPosition="Bottom" />
    </div>`
  })
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Label template

You can use the [labelTemplate](#) tag directive to specify a custom template for the **Label** of the rating. The current value of the rating will be passed as the **value** property in the template context when building the content of the label. This allows you to include dynamic information about the rating in the template.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
        <input ej-s-rating id="rating" value="3" showLabel="true"
[labelTemplate]="labelTemplate" />
    </div>`
})

```

```

        <ng-template #labelTemplate let-data="">
            <span>{{data.value}} out of 5</span>
        </ng-template>
    </div>`
    })
    export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip in Angular Rating Component

The rating component supports tooltip to show additional information in rating items by setting the [showTooltip](#) property. If enabled, the tooltip appears when the user hovers over a rating item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" showTooltip="true" value="3"
    />
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Tooltip template

You can use the [tooltipTemplate](#) tag directive to specify a custom template for the tooltip of the rating. The current value of the rating will be passed as the `value` property in the template context when building the content of the tooltip. This allows you to include dynamic information about the rating in the template.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" showTooltip="true"
[tooltipTemplate]="tooltipTemplate" />
      <ng-template #tooltipTemplate let-data="">
        <div [ngSwitch]="data.value">
          <span *ngSwitchCase="1">Angry</span>
          <span *ngSwitchCase="2">Sad</span>
          <span *ngSwitchCase="3">Neutral</span>
          <span *ngSwitchCase="4">Good</span>
          <span *ngSwitchDefault>Happy</span>
        </div>
      </ng-template>
    </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Tooltip customization

You can customize the appearance of the tooltips using the `cssClass` property of the rating component and by defining the custom styles for tooltip elements like the below example.

You can find more information about customizing the appearance of the tooltip in the [Tooltip Customization](#) documentation.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
```

```

imports: [
    FormsModule, RatingModule
],
standalone: true,
selector: 'app-root',
template: `<!-- To Render Rating component. -->
    <div class="wrap">
        <input ejs-rating id="rating" value="3" showTooltip="true"
cssClass='customtooltip' />
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.wrap {
    margin: 50px auto;
    text-align: center;
}
/* To change the radius of the tooltip corners. */
.customtooltip.e-tooltip-wrap {
    border-radius: 3px;
}
/* To change the size of the tooltip content. */
.customtooltip.e-tooltip-wrap .e-tip-content {
    font-size: 14px;
}
/* To change the border color and width for tooltip. */
.customtooltip.e-tooltip-wrap.e-popup {
    border: 2px solid #000000;
}
/* To change the color for arrow of the tooltip. */
.customtooltip.e-tooltip-wrap .e-arrow-tip-inner.e-tip-bottom {
    border: 12px #9693
}
/* To change the top border color for arrow of the tooltip. */
.customtooltip.e-tooltip-wrap .e-arrow-tip-outer.e-tip-bottom {
    border-top: 9.5px solid #000000;
}

```

Selection in Angular Rating Component

The rating component allows users to rate something using a visual scale, and the selection state can be changed by the user clicking or tapping on the stars in the rating scale or through code. The rating component has a minimum value and a reset button, and provides customization options for the selected rating value and selection behavior.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" />
    </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Min value

You can use the [min](#) property of the rating component to set the minimum possible rating value the user can select. If you set the `min` property to 2, then you will not be able to select a rating lower than 2.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
```

```

selector: 'app-root',
template: `<!-- To Render Rating component. -->
    <div class="wrap">
        <input ej2-rating id="rating" min="2" />
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Single selection

You can use the [enableSingleSelection](#) property of the rating component to select only one item at a time. When the `enableSingleSelection` property is set to `true`, only the selected item will be considered to be in the selected state, while all other items will be unselected.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
        <input ej2-rating id="rating" value="3"
enableSingleSelection="true" />
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show or hide reset button

You can reset the rating value to its default by using the [allowReset](#) property of the rating component. When the `allowReset` property is set to `true`, a reset button will be shown that allows the user to reset the rating value to its default.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" allowReset="true"
    />
    </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Templates in Angular Rating Component

The rating component allows you to customize the appearance of the rating items using templates. You can use templates to specify a custom layout for the rating items, which can include any content you want. This allows you to create a more customized and interactive rating experience for the user.

The rating component supports below templates for item customization.

- [emptyTemplate](#)
- [fullTemplate](#)

Empty (unrated) symbol template

To customize the appearance of **unrated** items, you can use the `emptyTemplate` tag directive. It allows you to specify the desired custom content for the unrated items.

The `value` and `index` are available in the template context for accessing information about the un-rated item.

If the `fullTemplate` is not defined, the `emptyTemplate` will be used as the default for both rated and unrated items. You can apply custom styles to differentiate between the rated and unrated states of the items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3"
[emptyTemplate]="emptyTemplate" />

      <ng-template #emptyTemplate>
        <span class="custom-font sf-rating-heart"></span>
      </ng-template>
    </div>`
})
export class AppComponent { }
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container .custom-font {
/* To add the background color for the font icon. */
```



```

background: linear-gradient(to right, rgb(254, 87, 133, 255) var(--rating-
value), transparent var(--rating-value));
/* To clip the background to the icon (text) alone. */
background-clip: text;
-webkit-background-clip: text;
/* To make the background color visible instead of font color. */
-webkit-text-fill-color: transparent;
/* To provide a border for font icon. */
-webkit-text-stroke: 1px rgb(254, 87, 133, 255);
}
/* Represents the styles for icon */
@font-face {
  font-family: 'rating';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjluSfQAAAEoAAAAVmNtYXNlEudaAAABjAAAADhnbHlm4LiF
sgAAAcwAAAJsaGVhZCKCSVkaAADQAAAAANmhoZWEIUQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAAmb
G9jYQCMATYAAAHEAAAAACG1heHABDwCZAAABCAAAACBuYW1l75Kp8wAABDgAAAIzCg9zdDjyU90AAA
ZUAAAAANwABAAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAwABAAAAAQAA2T6Kh18PPPU
ACwQAAAAAAN+4AkeAAAAA37gCQAAAAAAD9APaAAAAACAACAAAAAAAAAAAAEAAAADAI0AAgAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wHnAgQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAAEAAAAAA
AAAgAAAAMAAAAUAAMAAQAAABQABAakAAAABAAEAAEAAOoc//8AAOcB//8AAAABAAQAAAACAAEAAAA
AAIwBNgABAAAAAPza9oAfAAAEw8WFR8PPw41Lx4jDwwvDw8GqAwMDAsKCgoJCAgIBwYGBQUEBAMC
AgEBAQECAwMEBQULFSMhOVJliOxTOSEdFg0IBQQDAwIBAQBEBagIDBAQFBQYGBwgICakKCgoLDAwMD
AwMDQwMDQwZGBgYfXUVFBIRCAgGBwKLCwwNDg4PEBAQEREREhEODg4ODg4NA8IGBwcICakJCgoKCw
sMCwwNDA0MDQ0ODQ0ODQ0ODQ0NDRUimCtEX26P/V5FKycjFhQNDQ0ODQ0ODQ0NDgwNDQwNCwwMCwo
LCgoJCAkIBwcGBQUEAwMCAQECBQYJCw4PERMKCgsMEQ8PDQ0LCwoICAYFBAMCAQEBAgIEBAUAAgAA
AAD9APFAAMAJAAANZMRIwEPAXUXDwwRMZcfBDCXPwo9AS8FPwsvCDc1Pwg1LwU1Pw01LwkHJT8EN
S8LIw8BDK2tAfKCCgQBAQEBGCERERITigkJKBAGIQc1Bx45k9sOBQgLDQsJBQMEAgIECQYCAQEBAW
4ECQgGBwMDAQEBAMDAwKCAQEDFgsFBAQDAwICAgQECgEBAQQKBwcGBQUEAwMBAQEBAUHCQUBQY
R/q0PCQQDAgEBAwMKDBUDBwYMCw0HB1oBhwHeAQUDA3YfCgQsOh0bHBovCQgbDP6KAQEfAwEBAQIB
AQMGCGoMBggICAUICQgLBQQEBAUDBgMHCAGMCACIBwYGBgUFCQQCBgIEDAKGBQYHCQkKCQgIBwsEA
gUDAgQEBAUFBgCHCACGBgYGCgkIBgICAQEBAUYxGRobDQ0MDQsiHjEEBAIEAQECAAAAEgDeAAEAAA
AAAAAAAAQAAAAEAAAAAAAAEABgABAAEAAAAAAAAIABwAHAAEAAAAAAAAAMABgAOAAEAAAAAAAAQABgAUAAE
AAAAAAAAUACwAaAAEAAAAAAAAAYABgAlAAEAAAAAAAAaALAArAAEAAAAAAAAsAEgBXAAMAAQQJAAAAAgBp
AAMAAQQJAAEADABrAAMAAQQJAAIADgB3AAMAAQQJAAAMADACFAAMAAQQJAAQADACRAAMAAQQJAAUAF
gCdAAMAAQQJAAAYADACzAAMAAQQJAAoAWAC/AAMAAQQJAAAsAJAEXIHJhdGluZ1JlZ3VsYXJyYXRpbm
dyYXRpbmdWZXJzaW9uIDEuMHJhdGluZ0Zvb2N0Z2VzZXJhdGVkIHVzaW5nIFN5bmNmdXNpb24gTWV
0cm8gU3R1ZGlvd3d3LnN5bmNmdXNpb24uY29tACAACgBhAHQAaQBuAGcAUgBlAGcAdQBsAGEAcgBy
AGEAdABpAG4AZwByAGEAdABpAG4AZwBwAGUAcgBzAGkAbwBuACAAMQAuADAACgBhAHQAaQBuAGcAR
gBvAG4AdAAgAGcAZQBwAGUAcgBhAHQAQZQBkACAADQBzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG
8AbgAgAE0AZQB0AHIAbwAgAFMAdAB1AGQAaQBVAhCAdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgA
uAGMAbwBtAAAAAIAAAAAAAAAACgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAwECAQMBAAFaGvHcnQF
dGh1bWIAAAA=) format('truetype');
  font-weight: normal;
  font-style: normal;
}
[class^="sf-rating-"], [class*=" sf-rating-"] {
  font-family: 'rating' !important;
  speak: none;
  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-rating-heart:before {
  content: "\e702";
}

```

The current value of the rating item available in the template context as `value` and in the rating item element as CSS Variable (`--rating-value`) can be used to support precision in templates.

Full (rated) symbol template

To customize the appearance of **rated** items in the rating component, you can use the `fullTemplate` tag directive. This directive allows you to specify a custom layout for the rated items, which can include any content you desire.

The `value` and `index` are available in the template context for accessing information about the rated item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3"
[emptyTemplate]="emptyTemplate" [fullTemplate]="fullTemplate" />
      <ng-template #emptyTemplate>
        <span class="custom-font sf-icon-empty-star"></span>
      </ng-template>
      <ng-template #fullTemplate>
        <span class="custom-font sf-icon-fill-star"></span>
      </ng-template>
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container .custom-font {
  /* To change the icon font color. */
  color: rgb(255, 215, 0);
}
/* Represents the styles for icon */
@font-face {
  font-family: 'rating-template';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAKAIAAAwAgTlMvMjltSfMAAAEoAAAAVmNtYXNlEODaAAABjAAAADhnbHlm+icD
jQAAAcwAAAE0aGVhZCK49ucAAADQAAAAANmhoZWEIUQQEAAAArAAAACRobXR4DAAAAAAAAAYAAAAmB
G9jYQAcAJAAAHEAAAACG1heHABDwBkAAABCAAAACBuYwllmYExxgAAAwAAAAKfCg9zdCH169QAAA
WIAAAQAABAAAEAAAAAFwEAAAAAAD9AABAAAAAAAAAAAAAAAAAAAAAwABAAAAQAAGPX4jF8PPPU
ACwQAAAAAN/TWPsAAAAA39NY+wAAAAAD9AP0AAAACAACAAAAAAAAAAAAEAAAADAFgAAgAAAAAAgAA
AAoACgAAAP8AAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wDnAQQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAQA AAAEAAAAA
AAAgAAAAAMAAAUAMAAQAABQABAaKAAAABAAEAAEAAOCB//8AAOCa//8AAAABAAQAAAABAAIAAAA
AABwAmgABAAAAAAP0A/QACQAAAQUTAYUFAXMlAwFn/qX6OwE1ATU7+v6lmQKsNf7//pasrAFqAQE1
AUgAAAIAAAAAA/QD5AAAdAFcAAAEfBAUPAxUTLwEjDwETNS8DJT8EJwMFDwQVHwIDBx8EMzclBRczP
wU1Az8CNS8DJQMvBisBDwUCYAICBgMHASCwBAMCGuoHCAjPggIDBLEBHgcGBGJiHXb+uQgHBgQBAG
TUHgcECBAUHCAkIAQ4BDgcJBAQEbwQDHDMEAgMFBGf+tHYDAGMEBAQEBAQEBAQEAWICnwMDBgIDNBa
GBgYE/uCBAGKBAR0HBgYGSdQCBAYD3Fr+9jwDBAcHBQgIB9T+twUIBwcEAWKVlQIBAGIFBwgJAUnU
BwgJCACFBD0BCgQDBAICAgEBAGICBAMAAAAAEgDeAAEAAAAAAAAAAAAQAAAAEAAAAAAAAEADwABAAEAA
AAAAAIABwAQAAEAAAAAAAAAMADwAXAAEAAAAAAAAAQADwAmAAEAAAAAAAAUACwA1AAEAAAAAAAAAYADwBAAA
EAAAAAAALABPAAEAAAAAAAsAEgB7AAMAAQQJAAAAAGCNAAMAAQQJAAEAHGC PAAMAAQQJAAIADGc
tAAMAAQQJAAAMAHgC7AAMAAQQJAAQAHgDZAAMAAQQJAAUAFgD3AAMAAQQJAAAYAHgENAAMAAQQJAAoA
WAERAAAMAAQQJAAsAJAGDIHJhdGluZy10ZW1wbGF0ZVZlZ3VsYXJyYXRpbmctdGVtcGxhdGVyYXRpb
mctdGVtcGxhdGVWZXJzaW9uIDEuMHJhdGluZy10ZW1wbGF0ZUZvb2N0ZU9tY29tACAACgBhAHQAaQBwAGcALQB
0AGUAbQBwAGwAYQB0AGUAbQBwAGcAdQBtAGEAcgByAGEAdABpAG4AZwAtAHQAZQBtAHAAbABhAHQA
ZQByAGEAdABpAG4AZwAtAHQAZQBtAHAAbABhAHQA ZQBWAGUAcgBzAGkAbwBuACAAMQAuADAACgBhA
HQAAQBwAGcALQB0AGUAbQBwAGwAYQB0AGUArgBvAG4AdAAgAGcAZQBwAGUAcgBhAHQA ZQBkACAADQ
BzAGkAbgBnACAAUwB5AG4AYwBmAHUAcwBpAG8AbgAgAE0AZQB0AHIAbwAgAFMAdAB1AGQAaQBvAHc
AdwB3AC4AcwB5AG4AYwBmAHUAcwBpAG8AbgAuAGMABwBtAAAAAIAAAAAAAAAACgAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAwECAQMBBAAJZmlsbC1zdGFyCmVtcHR5LXN0YXIAAA==)
format('trueType');
  font-weight: normal;
  font-style: normal;
}
[class^="sf-icon-"],
[class*=" sf-icon-"] {
  font-family: 'rating-template' !important;
  speak: none;
  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-fill-star:before {
  content: "\e700";
}
.sf-icon-empty-star:before {
  content: "\e701";
}

```

Using Emoji icon as rating symbol

You can use emojis of your choice as rating symbol by specifying them as template content within the `emptyTemplate` tag directive.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" [emptyTemplate]="template"
enableAnimation="false" enableSingleSelection="true" value="3"/>
      <ng-template #template let-data="">
        <div [ngSwitch]="data.index">
          <span *ngSwitchCase="0" class="angry
emoji">😡</span>
          <span *ngSwitchCase="1" class="disagree
emoji">😞</span>
          <span *ngSwitchCase="2" class="neutral
emoji">😐</span>
          <span *ngSwitchCase="3" class="agree
emoji">😄</span>
          <span *ngSwitchDefault class="happy
emoji">😊</span>
        </div>
      </ng-template>
    </div>`
})
export class AppComponent { }

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
/* To change the color of an unselected rating item. */
.e-rating-item-container:not(.e-rating-selected) .emoji {
  filter: grayscale(1);
}
```

Using SVG icon as rating symbol

You can use SVG icons of your choice as rating symbol by specifying them as template content within the `emptyTemplate` and `fullTemplate` tag directives.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';

@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="4"
[emptyTemplate]="emptyTemplate" [fullTemplate]="fullTemplate"
enableAnimation="false" />
      <ng-template #emptyTemplate let-data="">
        <svg width="35" height="25" class="e-rating-svg-icon">
          <rect width="35" height="25" fill="transparent"
style="stroke-width:2;stroke:rgb(173,181,189)" />
        </svg>
```

```

        </ng-template>
        <ng-template #fullTemplate let-data="">
            <svg width="35" height="25" class="e-rating-svg-icon">
                <defs>
                    <linearGradient id="grad{{data.index}}" x1="0%"
y1="0%" x2="100%" y2="0%">
                        <stop class="start" offset="0%" />
                        <stop class="end" offset="100%" />
                    </linearGradient>
                </defs>
                <rect width="35" height="25"
style="fill:url(#grad{{data.index}});stroke-
width:2;stroke:rgb(173,181,189)"/>
            </svg>
        </ng-template>
    </div>`
    })
    export class AppComponent { }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
    color: #008cff;
    height: 40px;
    left: 45%;
    position: absolute;
    top: 45%;
    width: 30%;
}
.wrap {
    margin: 50px auto;
    text-align: center;
}
/* To change the size between items */
.e-rating-container .e-rating-item-container {
    padding: 0px;
}
/* To set the gradient color */
.e-rating-svg-icon #grad0 .start {
    stop-color: #FF0000;
}
.e-rating-svg-icon #grad0 .end,
.e-rating-svg-icon #grad1 .start {
    stop-color: #ff5101;
}
.e-rating-svg-icon #grad1 .end,
.e-rating-svg-icon #grad2 .start {
    stop-color: #ffc801;
}

```

```

}
.e-rating-svg-icon #grad2 .end,
.e-rating-svg-icon #grad3 .start {
  stop-color: #dbe300;
}
.e-rating-svg-icon #grad3 .end,
.e-rating-svg-icon #grad4 .start {
  stop-color: #8bc301;
}
.e-rating-svg-icon #grad4 .end {
  stop-color: #4eaa01;
}

```

Using PNG image as rating symbol

You can use PNG images of your choice as rating symbol by specifying them as template content within the `emptyTemplate` and `fullTemplate` tag directives.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3"
[emptyTemplate]="emptyTemplate" [fullTemplate]="fullTemplate" />
      <ng-template #emptyTemplate>
        
      </ng-template>
      <ng-template #fullTemplate>
        
      </ng-template>
    </div>`
})
export class AppComponent {}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Events in Angular Rating Component

This section describes the rating events that will be triggered when appropriate actions are performed. The following events are available in the rating component.

beforeItemRender

The rating component triggers the [beforeItemRender](#) event before rendering each rating item. The [RatingItemEventArgs](#) passed as an event argument provides the details of the item to be rendered.

`typescript

```
import { Component } from '@angular/core';
import { RatingItemEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component({
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
<div class="wrap">
<input ej2-rating id='rating' (beforeItemRender)="beforeItemRender($event)"/><br />
</div>`
})
export class AppComponent {
  public beforeItemRender(args: RatingItemEventArgs){
    //Your required action here
  };
}
```

created

The rating component triggers the [created](#) event when the rendering of the rating component is completed.

`typescript

```
import { Component } from '@angular/core';
import { RatingItemEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component({
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
<div class="wrap">
<input ej2-rating id='rating' (created)="created()"/><br />
</div>`
})
```



```
export class AppComponent {
  public created(){
    //Your required action here
  };
}
```

onItemHover

The rating component triggers the [onItemHover](#) event when the rating item is hovered. The [RatingHoverEventArgs](#) passed as an event argument provides the details of the hovered item.

`typescript

```
import { Component } from '@angular/core';
import { RatingHoverEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component({
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
<div class="wrap">
<input ej2-rating id='rating' (onItemHover)="onItemHover($event)"/><br />
</div>`
})
export class AppComponent {
  public onItemHover(args: RatingHoverEventArgs) {
    //Your required action here
  };
}
```

valueChanged

The rating component triggers the [valueChanged](#) event when the value of the rating is changed. The [RatingChangedEventArgs](#) passed as an event argument provides the details when value is changed.

`typescript

```
import { Component } from '@angular/core';
import { RatingChangedEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component({
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
```

```

<div class="wrap">
<input ejs-rating id='rating' (valueChanged)="valueChanged($event)"/><br />
</div>`
})
export class AppComponent {
public valueChanged(args: RatingChangedEventArgs) {
//Your required action here
};
}
`

```

Below example demonstrates the valueChanged event of the Rating component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
import { RatingChangedEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component({
imports: [
FormsModule, RatingModule
],
standalone: true,
selector: 'app-root',
template: `<!-- To Render Rating component. -->
<div class="wrap">
<input ejs-rating id='rating'
(valueChanged)="valueChanged($event)"/><br />
</div>`
})
export class AppComponent {
public valueChanged(args: RatingChangedEventArgs) {
alert("Previous Value:" + args.previousValue + "\nValue:" + args.value);
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Appearance in Angular Rating Component

You can also customize the appearance of rating component.

Items count

You can specify the number of rating items using the [itemsCount](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" itemsCount="8" value="3"/>
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Disabled

You can disable the rating component by using the [disabled](#) property. When the `disabled` property is set to `true`, the rating component will be disabled and the user will not be able to interact with it and a disabled rating component may have a different visual appearance than an enabled one.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
```

```

        <input ej2-rating id="rating" value="3" disabled="true"
    />
    </div>`
  })
  export class AppComponent {}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Visible

You can use the [visible](#) property of the rating component to component the visibility of the component. When the `visible` property is set to `true`, the rating component will be visible on the page. When it is set to `false`, the component will be hidden.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { RatingComponent } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <button id="btn" (click)="visible()">Visible</button>
      <input ej2-rating #rating id="rating" value="3"
visible="true" />
    </div>`
})
export class AppComponent {
  @ViewChild('rating')
  public rating?: RatingComponent;
  public visible() {
    this.rating!.visible = !this.rating?.visible;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Read only

You can use the [readOnly](#) property of the rating component to make the component non-interactive and prevent the user from changing the rating value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" readOnly="true"
    />
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

CssClass

You can customize the appearance of the rating component, such as by changing its colors, fonts, sizes, or other visual aspects by using the [cssClass](#) property.

Changing rating symbol border color

You can change the rating icon border color in rating component, you can use the `cssClass` property and set the `text-stroke` CSS property of `.e-rating-icon` to your desired border color.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" cssClass="custom-
border" />
    </div>`
  })
export class AppComponent {}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container.custom-border .e-rating-item-list:hover .e-rating-item-
container .e-rating-icon,
.e-rating-container.custom-border .e-rating-item-container .e-rating-icon {
  /* To change rating symbol border color */
  -webkit-text-stroke: 2px #ae9e9d;
}

```

Changing rated/un-rated symbol fill color

You can customize the fill colors of rated and un-rated icons in the rating component using the `cssClass` property and the `linear-gradient` color-stops in the `background` CSS property of `.e-rating-icon`. The **first** color-stop defines the rated fill color and the **second** defines the un-rated fill color.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule

```

```

    ],
    standalone: true,
    selector: 'app-root',
    template: `<!-- To Render Rating component. -->
      <div class="wrap">
        <input ejs-rating id="rating" value="3" cssClass="custom-
fill" />
      </div>`
  })
  export class AppComponent {}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

INDEX.CSS

```

/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container.custom-fill .e-rating-item-list:hover .e-rating-item-
container .e-rating-icon,
.e-rating-container.custom-fill .e-rating-item-container .e-rating-icon {
  /* To change rated symbol fill color and un-rated symbol fill color */
  background: linear-gradient(to right, #ffe814 var(--rating-value),
#d8d7d4 var(--rating-value));
  background-clip: text;
  -webkit-background-clip: text;
}

```

This will customize the rated fill color to `#ffe814` and un-rated fill color to `#d8d7d4`. `--rating-value` in the linear-gradient provides the current value of the rating item.

Changing the item spacing

You can change the space between the rating items in rating component, by using the `cssClass` property and setting the `margin` / `padding` CSS property of `.e-rating-item-container` to your desired size.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'

```

```
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ej2-rating id="rating" value="3" cssClass="custom-
font" />
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
.e-rating-container.custom-font .e-rating-item-container {
  /* To change the size between items */
  margin: 0px 7px;
}
```

Changing icon using CssClass

You can change the rating item icon in rating component, you can use the `cssClass` property and set the content CSS property of `.e-icons.e-star-filled:before` to your desired font icon.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { enableRipple } from '@syncfusion/ej2-base'
```



```
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, RatingModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To Render Rating component. -->
    <div class="wrap">
      <input ejs-rating id="rating" value="3" cssClass="custom-
icon" />
    </div>`
})
export class AppComponent {}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

INDEX.CSS

```
/* Represents the styles for loader */
#loader {
  color: #008cff;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
.wrap {
  margin: 50px auto;
  text-align: center;
}
/* Represents the styles for icon */
@font-face {
  font-family: 'custom-icon';
  src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgTlMvMjlvSfQAAAEoAAAAVmNtYXNlEudXAAABiAAAADZnbHlmVIZr
owAAAcgAAAEYAGVhZCK6KOUAAADQAAAAANmhoZWEIUAQDAAAArAAAACRobXR4CAAAAAAAYAAAAAIb
G9jYQCMAAAAAHAAAAABmlheHABDQCJAAABCAAAACBuYwllv3dY+QAAAUAAAAJVcG9zdN12YnkAAA
U4AAALWABAAAEAAAAAFwEAAAAAAD8wABAAAAAAAAAAAAAAAAAAAAAGABAAAAAQAAEGWKhV8PPPU
ACwQAAAAAAN/UcgCAAAAAA39RyBwAAAAAD8wPaAAAAACAACAAAAAEEAAAAACAH0AAQAAAAAAAgAA
AAoACgAAAP8AAAAAAAAAAQAAZAABQAAAokCzAAAAI8CiQLMAAAB6wAyAQgAAAIABQMAAAAAAAAAA
AAAAAAAAAAAAAAAAAUGZFZABA5wLnAgQAAAAAXAQAAAAAAAAABAAAAAAAAABAAAAAQAAAAAAACAA
AAAwAAABQAAwABAAAAFAAEACIAAAAEAAQAAQAA5wL//wAA5wL//wAAAAEABAAAAAEAAAAAAAAjAA
AAAEAAAAAA/MD2gB8AAATDxYVHw8/DjUvHiMPDC8PDwaoDAwMCwoKCgkICAgHbGyYFBQQEAWICAQEB
AQIDAQWQFBQsVIyE5UmWI7FM5IR0WDQgFBAMDagEBAQEACgMEBAUFBgYHCAGICQoKCgsMDAwMDAwND
AwNDBkYGBgXFRUUEhEICAYHCQsLDA0ODg8QEBARERESEQ4ODg4ODg0DwgYHBwgICQkKCgoLCwwLDA
0MDQwNDQ4NDQ4NDQ4NDQ0NFSIwK0Rfbo/9XkUrJyMWFA0NDQ4NDQ4NDQ0ODA0NDA0LDAwLCgsKCgk
ICQgHbWYFBQQDAWIBAQIFBgkLDg8REwoKCwwRDw8NDQsLCggIBgUEAwIBAQEACgQEBQAAABIA3gAB
AAAAAAAAAAEAAAABAAAAAABAAsAAQABAAAAAAACAacADAABAAAAAADAAsAEwABAAAAAAAEAAAsAH
gABAAAAAAAFaAsAKQABAAAAAAGAAsANAABAAAAAAAKACwAPwABAAAAAALABIAawADAAEECQAAAA
```

```

IAfQADAAEECQABABYAfwADAAEECQACAA4AlQADAAEECQADABYAowADAAEECQAEABYAuQADAAEECQA
FABYAzwADAAEECQAGABYA5QADAAEECQAKAFgA+wADAAEECQALACQBUyBjdXN0b20taWNvblJlZ3Vs
YXJjdXN0b20taWNvbmNlc3RvbSlpY29uVmVyc2lubiAxLjBjdXN0b20taWNvbklzbnQgZ2VuZXJhd
GVkIHVzaW5nIFN5bmNmdXNpb24gTWV0cm8gU3RlZGlvZ3d3LnN5bmNmdXNpb24uY29tACAAYwB1AH
MAdABvAG0ALQBpAGMAbwBuAFIAZQBnAHUAbABhAHIAyWb1AHMAdABvAG0ALQBpAGMAbwBuAGMAQB
zAHQAbwBtAC0AaQBjAG8AbgBWAGUAcgBzAGkAbwBuACAAMQAUADAAYwB1AHMAdABvAG0ALQBpAGMA
bwBuAEYAbwBuAHQAIABnAGUAbgB1AHIAyQB0AGUAZAAGAHUAcwBpAG4AZwAgAFMAeQBvAGMAZgB1A
HMAaQBvAG4AIABNAGUAdABYAG8AIABTAHQAdQBkAGkAbwB3AHcAdwAuAHMAeQBvAGMAZgB1AHMAaQ
BvAG4ALgBjAG8AbQAAAAACAAAAAAAAAaAAAAAIAAgEDAAVoZWZ
ydAAAAA==) format('trueType');
  font-weight: normal;
  font-style: normal;
}
.custom-icon .e-icons.e-star-filled {
  font-family: 'custom-icon' !important;
  speak: none;
  font-style: normal;
  font-weight: normal;
  font-variant: normal;
  text-transform: none;
  line-height: 1;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}
.custom-icon .e-icons.e-star-filled:before {
  content: "\e702";
}

```

Accessibility in Angular Rating component

The Rating 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 Rating 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 Rating component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Rating component:

| Attributes | Purpose |

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

| **role=slider** | It defines an input where the user selects a value from within a specified range. |

| **role=button** | Specifies that the reset is a clickable element that resets the rating to its minimum value. |

| **aria-label** | Provides an accessible name for Rating. |

| **aria-valuemin** | It defines the minimum value of rating. |

| **aria-valuemax** | It defines the maximum value of rating. |

| **aria-valuenow** | It defines the current value of rating. |

| **aria-hidden** | It specifies whether the reset button is interactive or not. |

Keyboard interaction

The Rating 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 Rating component.

| Keyboard shortcuts | Actions |

```
|-----|-----|
| Space | When Reset Button is focused, resets to min value. |
| Arrow Up | Increases the value. |
| Arrow Left | Decreases the value; in RTL mode, increases the value. |
| Arrow Down | Decreases the value. |
| Arrow Right | Increases the value; in RTL mode, decreases the value. |
```

Ensuring accessibility

The Rating component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Rating component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Rating component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Ribbon

Getting started with Angular Ribbon component

This section explains how to create a simple Ribbon, and demonstrate the basic usage of the Ribbon module in an Angular environment.

Dependencies

The list of dependencies required to use the Ribbon module in your application is given below:

```
`javascript
|-- @syncfusion/ej2-angular-ribbon
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-ribbon
\`
```

Setup Angular environment

You can use [Angular CLI](#) to setup your Angular applications. To install Angular CLI use the following command.

`

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

`

Create an Angular application

Start a new Angular application using below Angular CLI command.

`

```
ng new my-app
```

```
cd my-app
```

`

Installing Syncfusion Ribbon Package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(`>=20.2.36`) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-ribbon](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-ribbon --save
```

`

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the ngcc package use the below.

Add [@syncfusion/ej2-angular-ribbon@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-ribbon@ngcc --save
```

`

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
```

```
@syncfusion/ej2-angular-ribbon:"21.1.35-ngcc"
```

```
,
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Ribbon module

Import Ribbon module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-ribbon`.

```
`javascript
import { NgModule } from "@angular/core";
import { BrowserModule } from "@angular/platform-browser";
// Import Syncfusion Ribbon module from ribbon package.
import { RibbonModule, RibbonAllModule } from "@syncfusion/ej2-angular-ribbon";
import { AppComponent } from "./app.component";
@NgModule({
  imports: [BrowserModule, RibbonModule, RibbonAllModule], // Registering EJ2 Ribbon Module.
  declarations: [AppComponent],
  bootstrap: [AppComponent],
})
export class AppModule {}
`,`
```

Adding CSS reference

Add Ribbon component's styles as given below in `style.css`.

```
`css
@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";
@import "../node_modules/@syncfusion/ej2-lists/styles/material.css";
@import "../node_modules/@syncfusion/ej2-dropdowns/styles/material.css";
@import "../node_modules/@syncfusion/ej2-navigations/styles/material.css";
@import "../node_modules/@syncfusion/ej2-ribbon/styles/material.css";
@import "../node_modules/@syncfusion/ej2-angular-ribbon/styles/material.css";
`,`
```

Adding Syncfusion Ribbon component

Modify the template in `app.component.ts` file with `ejs-ribbon` to render the Ribbon component.

```
`javascript
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon"></ejs-ribbon>`,
})
export class AppComponent {}
`
```

Adding Ribbon Tab

In Ribbon, the options are arranged in tabs for easy access. You can use the `<e-ribbon-tab>` selector to define the ribbon tab like below.

```
`javascript
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home"></e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {}
`
```

Adding Ribbon Group

To define a ribbon group under each tab, you can use the `<e-ribbon-group>` selector like below. The `orientation` property of ribbon group defines whether the collection of items will be rendered column-wise or row-wise.

```
`javascript
import { Component } from "@angular/core";
@Component({
  selector: "app-root",
```

```

template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" orientation="Row"></e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {}
`

```

Adding Ribbon Item

You can use the `<e-ribbon-collection>` selector to define each ribbon collection that contains one or more items. To define each ribbon item, you can use the `<e-ribbon-item>` selector and the `type` property to specify the type of component to be rendered, like a button, a drop-down button, a combo box, and more.

```

`javascript
import { Component } from "@angular/core";

import { RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" orientation="Row">
<e-ribbon-collections>
<e-ribbon-collection id="paste-collection">
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>

```



```

</e-ribbon-collection>
<e-ribbon-collection id="cutcopy-collection">
  <e-ribbon-items>
    <e-ribbon-item type="Button" [buttonSettings]="cutButton">
  </e-ribbon-item>
    <e-ribbon-item type="Button" [buttonSettings]="copyButton">
  </e-ribbon-item>
  </e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste" };
    public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
    public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  }
},

```

Running the application

Run the application in the browser using the following command:

```

ng serve

```

The following example illustrates how tabs, groups, collections, and items are used in a ribbon component to form the ribbon layout.

APP.COMPONENT.TS

```

import { Component } from '@angular/core';
import { FileMenuSettingsModel, RibbonButtonSettingsModel,
RibbonSplitButtonSettingsModel, RibbonComboBoxSettingsModel,
RibbonDropDownSettingsModel, RibbonItemSize, RibbonCheckBoxSettingsModel,

```

```

RibbonColorPickerSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService, RibbonColorPickerService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste" };
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" } ] };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold", content: "Bold" };
  public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-italic", content: "Italic" };
  public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-underline", content: "Underline" };
  public strikethroughButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-strikethrough", content: "Strikethrough" };
  public changecaseButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-change-case", content: "Change Case" };
  public editorButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-edit", content: "Editor" };
  public chartButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-chart", content: "Chart" };
  public printButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-print-layout", content: "Print Layout" };
  public webButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-web-layout", content: "Web Layout" };
  public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18", "20", "22", "24", "26", "28", "36", "48", "72", "96"];
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"];
  public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width: "150px", allowFiltering: true };
  public fontsizeSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontSize, index: 3, width: "65px", allowFiltering: true };
  public colorSettings: RibbonColorPickerSettingsModel = { value: "#123456" };
  public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: false };
  public grid: RibbonCheckBoxSettingsModel = { label: "Gridlines", checked: false };
}

```

```

public navigation: RibbonCheckBoxSettingsModel = { label: "Navigation
Pane", checked: false };
public fileSettings: FileMenuSettingsModel = {
  menuItems: [
    { text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" }
  ],
  visible: true
};
public largeSize: RibbonItemSize = RibbonItemSize.Large;
public smallSize: RibbonItemSize = RibbonItemSize.Small;
public Simplified: DisplayMode = DisplayMode.Simplified;
public Overflow: DisplayMode = DisplayMode.Overflow;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="default" [fileMenu]="fileSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" id="clipboard" groupIconCss="e-
icons e-paste" [showLauncherIcon]=true>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton" [allowedSizes]="largeSize"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button" [buttonSettings]="cutButton">
            </e-ribbon-item>
              <e-ribbon-item type="Button" [buttonSettings]="copyButton">
            </e-ribbon-item>
              <e-ribbon-item type="Button" [buttonSettings]="formatButton">
            </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
      <e-ribbon-group header="Font" orientation="Row"
[enableGroupOverflow]=true [isCollapsible]=false
      groupIconCss="e-icons e-bold" cssClass="font-group">
        <e-ribbon-collections>

```

```

        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontstyleSettings">
            </e-ribbon-item>
                <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontsizeSettings">
            </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="ColorPicker" [allowedSizes]="smallSize"
[displayOptions]='Simplified'
                [colorPickerSettings]="colorSettings">
            </e-ribbon-item>
                <e-ribbon-item type="Button" [allowedSizes]="smallSize"
[buttonSettings]="boldButton">
            </e-ribbon-item>
                <e-ribbon-item type="Button" [allowedSizes]="smallSize"
[buttonSettings]="italicButton">
            </e-ribbon-item>
                <e-ribbon-item type="Button" [allowedSizes]="smallSize"
[buttonSettings]="underlineButton">
            </e-ribbon-item>
                <e-ribbon-item type="Button" [allowedSizes]="smallSize"
[buttonSettings]="strikethroughButton">
            </e-ribbon-item>
                <e-ribbon-item type="Button" [allowedSizes]="smallSize"
[buttonSettings]="changeCaseButton">
            </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
    <e-ribbon-group header="Editor" [isCollapsible]=false
groupIconCss="e-icons e-edit">
        <e-ribbon-collections>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="Button" [allowedSizes]="largeSize"
[buttonSettings]="editorButton">
                </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
<e-ribbon-tab header="Insert">
    <e-ribbon-groups>
        <e-ribbon-group header="tables" [isCollapsible]="false">
            <e-ribbon-collections>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item type="DropDown" [allowedSizes]="largeSize"
[dropDownSettings]="tableSettings">

```

```

        </e-ribbon-item>
      </e-ribbon-items>
    </e-ribbon-collection>
  </e-ribbon-collections>
</e-ribbon-group>
<e-ribbon-group header="Illustrations" id="illustration"
[showLauncherIcon]=true orientation="Row"
[enableGroupOverflow]=true>
  <e-ribbon-collections>
    <e-ribbon-collection>
      <e-ribbon-items>
        <e-ribbon-item [allowedSizes]="largeSize"
[buttonSettings]="chartButton">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
<e-ribbon-group header="Media" [isCollapsible]="false">
  <e-ribbon-collections>
    <e-ribbon-collection>
      <e-ribbon-items>
        <e-ribbon-item type="Template" [itemTemplate]="itemTemplate">
          <ng-template #itemTemplate let-data="">
            <span class="ribbonTemplate {{data.activeSize}}">
              <span class="e-icons e-video"></span>
              <span class="text">Video</span>
            </span>
          </ng-template>
        </e-ribbon-item>
      </e-ribbon-items>
    </e-ribbon-collection>
  </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
<e-ribbon-tab header="View">
  <e-ribbon-groups>
    <e-ribbon-group header="Views" orientation="Row" groupIconCss="e-
icons e-print">
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item [allowedSizes]="largeSize" type="Button"
[buttonSettings]="printButton">
              </e-ribbon-item>
            <e-ribbon-item [allowedSizes]="largeSize" type="Button"
[buttonSettings]="webButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    <e-ribbon-group header="show" [isCollapsible]=false>
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>

```

```

        <e-ribbon-item type="CheckBox" [checkBoxSettings]="ruler">
        </e-ribbon-item>
        <e-ribbon-item type="CheckBox" [checkBoxSettings]="grid">
        </e-ribbon-item>
        <e-ribbon-item type="CheckBox"
[checkBoxSettings]="navigation">
        </e-ribbon-item>
        </e-ribbon-items>
        </e-ribbon-collection>
        </e-ribbon-collections>
        </e-ribbon-group>
        </e-ribbon-groups>
        </e-ribbon-tab>
        </e-ribbon-tabs>
    </ejs-ribbon>

```

APP.COMPONENT.CSS

```

/* Represents the styles for ribbonTemplate */
.ribbonTemplate {
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
}
.ribbonTemplate.Large {
    flex-direction: column;
}
.ribbonTemplate.Large .e-icons {
    font-size: 35px;
}
.ribbonTemplate.Medium .e-icons,
.ribbonTemplate.Small .e-icons{
    font-size: 20px;
    margin: 15px 5px;
}
.ribbonTemplate.Small .text {
    display:none;
}
/* Represents the styles for Ribbon group */
.font-group .e-ribbon-group-content {
    justify-content: center;
}

```

Modules in Ribbon component

The following modules are available in Ribbon. If the module injection type is **selective**, manual injection is required to extend the Ribbon's functionality.

Module	Description	Module Injection Type
-----	-----	-----
RibbonButtonService	To use the built-in button as a ribbon item.	default
RibbonCheckBoxService	To use the built-in checkbox as a ribbon item.	default

- | `RibbonDropDownService` | To use the built-in dropdown button as a ribbon item. | default |
- | `RibbonSplitButtonService` | To use the built-in split button as a ribbon item. | default |
- | `RibbonComboBoxService` | To use the built-in combobox as a ribbon item. | default |
- | `RibbonGroupButtonService` | To use the built-in groupbutton as a ribbon item. | default |
- | `RibbonColorPickerService` | Inject this module to use the built-in colorpicker as a ribbon item. | selective |
- | `RibbonGalleryService` | Inject this module to use the gallery as a ribbon item. | selective |
- | `RibbonFileMenuService` | Inject this module to use the file menu feature. | selective |
- | `RibbonBackstageService` | Inject this module to use the backstage view feature. | selective |
- | `RibbonContextualTabService` | Inject this module to use the contextual tab feature. | selective |
- | `RibbonKeyTipService` | Inject this module to use the keytip feature. | selective |

These modules should be injected into the `providers` section of root `NgModule` or component class.

Tabs and Groups

The Ribbon component consists of a series of tabs that are organized into groups to enable quick access to specific commands or tools. Each tab contains a set of groups, and each group contains collections of items that are logically related to each other.

Adding Tabs

You can use the `tabs` property to add tabs to the Ribbon component and define the content of the tab header by using the `header` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
```

```

    <e-ribbon-tabs>
      <e-ribbon-tab header="Home">
      </e-ribbon-tab>
      <e-ribbon-tab header="Insert">
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>

```

Adding Groups

You can use the [groups](#) property to add groups for each tab in the Ribbon and define the name of the group header by using the [header](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
    <e-ribbon-tab header="Insert">
      <e-ribbon-groups>
        <e-ribbon-group header="Tables">
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```


Adding Items

You can add collections of items to each group by using the [collections](#) and [items](#) properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
          <e-ribbon-items>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-group>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>
```

```

        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

For more information on the built-in and how to add custom Ribbon items, you can visit the [items](#) page.

Items in Angular Ribbon component

Ribbon renders various built-in items based on the item [type](#) property. By default, the type property is set as **Button** which renders the Button.

Built-in items

You can render the built-in Ribbon items by using the `<e-ribbon-item>` tag directive, to specify the [type](#) property.

The following table explains the built-in items and their actions.

Built-in Ribbon Items	Actions
Button	Renders button as ribbon item.
CheckBox	Renders checkbox as ribbon item.
DropDown	Renders dropdownbutton as ribbon item.
SplitButton	Renders splitbutton as ribbon item.
ComboBox	Renders combobox as ribbon item.
ColorPicker	Renders color picker as ribbon item.
GroupButton	Renders groupbutton as ribbon item.

Button items

You can render the built-in button Ribbon item by setting the [type](#) property as **Button**. You can also customize the button item using the [RibbonButtonSettingsModel](#), which provides options such as **iconCss**, **content**, **isToggle** and more.

Toggle button

The [isToggle](#) property can be used to define whether the button act as a toggle button or not. By default, the value is **false**.

```
`javascript
```

```
import { Component } from "@angular/core";
```

```
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
```

```
@Component({
```

```
  selector: "app-root",
```

```
  template: `<!-- To Render Ribbon. -->
```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button" [buttonSettings]="cutButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut", isToggle: true
};
}
`

```

Checkbox items

You can render the built-in checkbox Ribbon item by setting the [type](#) property to `CheckBox`. You can also customize the checkbox item using the [RibbonCheckBoxSettingsModel](#), which provides options such as `labelPosition`, `label`, `checked` and more.

Checkbox state

You can use the [checked](#) property to handle the checked or unchecked state. By default, the value is `false`.

```

`javascript
import { Component } from "@angular/core";
import { RibbonCheckBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({

```

```

selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type=CheckBox [checkBoxSettings]="ruler">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: true };
}
`

```

Defining label

You can use the [label](#) property to add a caption for the CheckBox. The label position can be set **Before** or **After**, by using the [labelPosition](#) property. By default, the labelPosition is **After**.

```

`javascript
import { Component } from "@angular/core";
import { RibbonCheckBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type=CheckBox [checkBoxSettings]="ruler">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: true, labelPosition: "Before" };
}
`

```

DropDown button items

You can render the built-in dropDown Ribbon item by setting the [type](#) property to **DropDown**. You can also customize the dropDown item through [RibbonDropDownSettingsModel](#), which provides options such as **iconCss**, **content**, **target** and more.

Target

The [target](#) property specifies the element selector to be displayed in the DropDownButton popup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';

```

```
@Component({
  imports: [ RibbonModule, ListViewModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", target: "#tableList" };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="tables" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>
  <ejs-listview id='tableList' [dataSource]='tableOptions' headerTitle='Table'
[showHeader]='true'></ejs-listview>
```

Customize Dropdown button item

You can customize the dropdown button item by specifying a custom cssClass using the [beforeItemRender](#) event.

The following sample showcases how to customize a specific dropdown item.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
```

```

import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel, MenuEventArgs } from "@syncfusion/ej2-angular-splitbuttons";
@Component({
  imports: [ RibbonModule, ListViewModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, beforeItemRender: (args: MenuEventArgs) => {
    if (args.item.text === 'Insert Table') {
      args.element.classList.add("e-custom-class");
    }
  } };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Insert">
      <e-ribbon-groups>
        <e-ribbon-group header="Tables">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

APP.COMPONENT.CSS

```
.e-custom-class {
  color: green;
}
```

Create dropdown popup on demand

You can handle the creation of popups, by using the [createPopupOnClick](#) property. If set to `true`, the popup will only be created upon opening. By default the value is `false`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon';
import { ListViewModule } from '@syncfusion/ej2-angular-lists';
import { Component } from '@angular/core';
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule, ListViewModule ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, createPopupOnClick: true };
};
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Insert">
      <e-ribbon-groups>
        <e-ribbon-group header="Tables">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
```



```

        </e-ribbon-group>
    </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Split button items

You can render the built-in splitButton Ribbon item by setting the [type](#) property to `SplitButton`. You can also customize the splitButton item through [RibbonSplitButtonSettingsModel](#), which provides options such as `iconCss`, `items`, `target` and more.

Target

The [target](#) property specifies the element selector to be displayed in the SplitButton popup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule, ListViewModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-table", content: "Table", target: "#tableList" };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="tables" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>

```

```

                                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="tableSettings">
                                </e-ribbon-item>
                                </e-ribbon-items>
                                </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                </e-ribbon-groups>
                                </e-ribbon-tab>
                                </e-ribbon-tabs>
</ejs-ribbon>
<ejs-listview id='tableList' [dataSource]='tableOptions' headerTitle='Table'
[showHeader]='true'></ejs-listview>

```

Combobox items

You can render the built-in comboBox Ribbon item by setting the [type](#) property to `ComboBox`. You can also customize the comboBox item through [RibbonComboBoxSettingsModel](#), which provides options such as `allowFiltering`, `autoFill`, `index`, `sortOrder` and more.

Filtering

You can use the [allowFiltering](#) property to filter the data items. The filtering operation is initiated automatically, as soon as you start typing characters. If no match is found the value of the `noRecordsTemplate` property will be displayed. By default, the value is `false`.

```
`javascript
```

```

import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>

```

```

</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];

  public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3,
    allowFiltering: true };
}
`

```

Index

You can use the [index](#) property to get or set the selected item in the combobox.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
  <ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
  <e-ribbon-tab header="Home">
  <e-ribbon-groups>
  <e-ribbon-group header="Clipboard" >
  <e-ribbon-collections>
  <e-ribbon-collection>
  <e-ribbon-items>
  <e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
  </e-ribbon-item>
  </e-ribbon-items>

```

```

</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];
  public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3 };
}
`

```

SortOrder

You can use the [sortOrder](#) property to specify the order in which the DataSource should be sorted.

None	The data source is not sorted.
Ascending	The data source is sorted in ascending order.
Descending	The data source is sorted in descending order.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
  <ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
  <e-ribbon-tab header="Home">
  <e-ribbon-groups>
  <e-ribbon-group header="Clipboard" >
  <e-ribbon-collections>
  <e-ribbon-collection>

```

```

<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3,
sortOrder: "Descending" };
}
`

```

Colorpicker items

You can render the built-in colorPicker Ribbon item by setting the [type](#) property to `ColorPicker`. You can also customize the colorPicker item through [RibbonColorPickerSettingsModel](#), which provides options such as `value`, `columns`, `showButtons` and more.

Value

You can use the [value](#) property to specify the color value. The value should be specified as Hex code.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">

```

```

<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection id="paste-collection">
<e-ribbon-items>
<e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456' };
}
`

```

Groupbutton items

You can render the built-in groupbutton Ribbon item by setting the [type](#) property to `GroupButton`. You can also customize the groupbutton item using the [RibbonGroupButtonSettingsModel](#), which provides options such as `selection` and `items`.

Items

You can render the groupbutton items by using the `<e-ribbon-item>` tag directive. You can also customize the groupbutton items through [RibbonGroupButtonItemModel](#), which provides options such as `content`, `iconCss`, `selected` and more.

Item content

You can use the [content](#) property to define the text content for the groupbutton.

```

`javascript
import { Component } from "@angular/core";
import { RibbonItemSize, RibbonGroupButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",

```

```

templateUrl: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Paragraph" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="GroupButton" [allowedSizes]="mediumSize"
[groupButtonSettings]="groupButtonContent">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public groupButtonContent: RibbonGroupButtonSettingsModel = {
items: [
{iconCss: 'e-icons e-align-left', content: 'Align Left'},
{iconCss: 'e-icons e-align-center', content: 'Align Center'},
{iconCss: 'e-icons e-align-right', content: 'Align Right'},
{iconCss: 'e-icons e-justify', content: 'Justify'}
]
}
public mediumSize: RibbonItemSize = RibbonItemSize.Medium;
}

```

Icon only

You can use the [iconCss](#) property to customize the groupbutton icon. If the `iconCss` property is not defined, the groupbutton will not be rendered.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonItemSize, RibbonGroupButtonSettingsModel } from
 '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public groupButtonIcon: RibbonGroupButtonSettingsModel = {
    items: [
      {iconCss: 'e-icons e-align-left'},
      {iconCss: 'e-icons e-align-center'},
      {iconCss: 'e-icons e-align-right'},
      {iconCss: 'e-icons e-justify'}
    ]
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Paragraph" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="GroupButton"
[allowedSizes]="smallSize" [groupButtonSettings]="groupButtonIcon">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
```



```
</e-ribbon-tabs>
</ejs-ribbon>
```

Selection

You can use the [selected](#) property to select the groupbutton item initially. When set to **true**, the button will be selected. By default the **selected** property is false.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonItemSize, RibbonGroupButtonSettingsModel } from
 '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public groupButtonSelected: RibbonGroupButtonSettingsModel = {
    items: [
      {iconCss: 'e-icons e-align-left', content: 'Align Left'},
      {iconCss: 'e-icons e-align-center', content: 'Align Center', selected:
true},
      {iconCss: 'e-icons e-align-right', content: 'Align Right'},
      {iconCss: 'e-icons e-justify', content: 'Justify'}
    ]
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Paragraph" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="GroupButton"
[allowedSizes]="smallSize" [groupButtonSettings]="groupButtonSelected">
                </e-ribbon-item>
              </e-ribbon-items>
            </e-ribbon-collection>
          </e-ribbon-collections>
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>
```

```

        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Single selection

You can set the [selection](#) property value as `RibbonGroupButtonSelection.Single` to make one selection at a time. It automatically deselects the previous choice when a different item is clicked.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonItemSize, RibbonGroupButtonSettingsModel,
RibbonGroupButtonSelection } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public singleSelection: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      {iconCss: 'e-icons e-align-left', content: 'Align Left'},
      {iconCss: 'e-icons e-align-center', content: 'Align Center', selected:
true},
      {iconCss: 'e-icons e-align-right', content: 'Align Right'},
      {iconCss: 'e-icons e-justify', content: 'Justify'}
    ]
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Paragraph" >
          <e-ribbon-collections>

```

```

        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="GroupButton"
[allowedSizes]="smallSize" [groupButtonSettings]="singleSelection">
                    </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
        </e-ribbon-collections>
    </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Multiple selection

You can set the [selection](#) property value as `RibbonGroupButtonSelection.Multiple` to select more than one button at a time. Users can select a button one by one to select multiple buttons.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonItemSize, RibbonGroupButtonSettingsModel,
RibbonGroupButtonSelection } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public multipleSelection: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Multiple,
    items: [
      {iconCss: 'e-icons e-bold', content: 'Bold'},
      {iconCss: 'e-icons e-italic', content: 'Italic', selected: true},
      {iconCss: 'e-icons e-underline', content: 'Underline'},
      {iconCss: 'e-icons e-strikethrough', content: 'Strikethrough'},
    ],
    selected: true
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Paragraph" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="GroupButton"
[allowedSizes]="smallSize" [groupButtonSettings]="multipleSelection">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>

```

Groupbutton in simplified mode layout

In simplified mode, the groupbutton will be rendered as a dropdownbutton. The dropdownbutton icon will be updated based on the button item selected. The initial button icon will be the set, if none of the buttons are selected.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonItemSize, RibbonGroupButtonSettingsModel,
RibbonGroupButtonSelection } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public groupButton: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      {iconCss: 'e-icons e-align-left', content: 'Align Left'},
      {iconCss: 'e-icons e-align-center', content: 'Align Center', selected:
true},
      {iconCss: 'e-icons e-align-right', content: 'Align Right'},
      {iconCss: 'e-icons e-justify', content: 'Justify'}
    ]
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" activeLayout="Simplified">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Paragraph" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="GroupButton"
[allowedSizes]="smallSize" [groupButtonSettings]="groupButton">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>
```

Custom items

You can customize the ribbon items with non-built-in items or HTML content by setting the [type](#) property to **Template**. This provides an option to customize the ribbon items with greater flexibility.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Templates" [isCollapsible]=false>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Template"
[itemTemplate]="customTemplate">
                  <ng-template #customTemplate let-data="">
                    <span class="custom-template
{{data.activeSize}}">
                      <label for="fname">First
name:</label>
                      <input type="text" id="fname"
name="fname">
                      <br><br>
                      <label for="lname">Last name:</label>
name="lname">
                      <input type="text" id="lname"
                      </span>
                    </ng-template>
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
          <e-ribbon-group header="Multimedia" >
            <e-ribbon-collections>
              <e-ribbon-collection>
                <e-ribbon-items>
                  <e-ribbon-item type="Template"
[itemTemplate]="itemTemplate">
                    <ng-template #itemTemplate let-data="">
                      <span class="ribbonTemplate
{{data.activeSize}}">
                        <span class="e-icons e-video"></span>
                        <span class="text">Video</span>
                      </span>
                    </ng-template>
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>

```

APP.COMPONENT.CSS

```
@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';
/* Represents the styles for Custom Ribbon Item */
.ribbonTemplate {
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
}

.ribbonTemplate.Large {
    flex-direction: column;
}

.ribbonTemplate.Large .e-icons {
    font-size: 35px;
}

.ribbonTemplate.Medium .e-icons,
.ribbonTemplate.Small .e-icons{
    font-size: 20px;
    margin: 15px 5px;
}

.ribbonTemplate.Small .text {
    display:none;
}

.custom-template input {
    margin-left: 10px;
    width: 100px;
}

.custom-template.Medium {
    display: flex;
    align-items: center;
}

.custom-template.Medium input {
    height: 14px;
    margin-right: 10px;
}

```

Items display Mode

You can use the [displayOptions](#) property to display the items in the Ribbon layout.

Auto	The items are displayed in all layouts based on the ribbon's overflow state.
Classic	The items are displayed only in the classic layout group.

Simplified	The items are displayed only in the simplified layout group.
Overflow	The items are displayed only in the overflow popup.

Display items in Classic only

To display the items only in the classic layout group, set the mode as `DisplayMode.Classic` in the [displayOptions](#) property.

```
`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item [displayOptions]='buttonDisplayMode' type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
```



```
public buttonDisplayMode: DisplayMode = DisplayMode.Classic;
}
,
```

Display items in Simplified only

To display the items only in the simplified layout group, set the mode as `DisplayMode.Simplified` in the [displayOptions](#) property.

```
`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item [displayOptions]='buttonDisplayMode' type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public buttonDisplayMode: DisplayMode = DisplayMode.Simplified;
```

```
}
,
```

Display items in Overflow popup only

To display the items only in the overflow, set the mode as `DisplayMode.Overflow` in the [displayOptions](#) property.

```
`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item [displayOptions]='buttonDisplayMode' type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public buttonDisplayMode: DisplayMode = DisplayMode.Overflow;
}
```

Enable or disable items

You can use the [disabled](#) property to disable a Ribbon item. It prevents the user interaction when set to `true`. By default, the value is `false`.

`javascript

```
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" >
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item [disabled]="true" type="Button" [buttonSettings]="cutButton">
              </e-ribbon-item>
                <e-ribbon-item type=CheckBox [checkBoxSettings]="ruler">
              </e-ribbon-item>
                <e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
              </e-ribbon-item>
                <e-ribbon-item type="SplitButton" [splitButtonSettings]="tableSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
```

```

})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut", isToggle: true
};
  public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: true };
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table";
}
,

```

Layouts in Angular Ribbon component

The Ribbon allows to customize the layout by using the [activeLayout](#) property. The Ribbon component supports the following layouts:

Classic layout

In classic layout, the Ribbon component organizes the items and groups in a traditional form by setting the [activeLayout](#) property to [Classic](#). By default, the Ribbon component renders in the [Classic](#) layout.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel,RibbonDropDownSettingsModel } from
 '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from "@syncfusion/ej2-angular-splitbuttons";
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This
device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-
table", content: "Table", items: this.tableOptions };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
    <e-ribbon-tab header="Insert">
      <e-ribbon-groups>
        <e-ribbon-group header="Tables">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Defining items size

You can use the [allowedSizes](#) property to set the allowed size for an item. The Ribbon items can be appeared in three different sizes: Large(large icon with text), Medium(small icon with text) and Small(small icon only). On resizing, the items size can be changed based on the available width of the tab content from the order of Large-> Medium-> Small and vice versa.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
RibbonItemSize } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  public largeSize: RibbonItemSize = RibbonItemSize.Large;
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
  public mediumSize: RibbonItemSize = RibbonItemSize.Medium;

  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste", items: this.pasteOptions };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[allowedSizes]="largeSize" [splitButtonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[allowedSizes]="mediumSize" [buttonSettings]="cutButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

```

        <e-ribbon-item type="Button"
[allowedSizes]="smallSize" [buttonSettings]="copyButton">
            </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Defining items orientation

The Ribbon group [orientation](#) property allows to manage how the items are aligned either in a **Row** or **Column**. By default, the orientation is set to **Column**, in which the items are arranged vertically.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];

  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste", items: this.pasteOptions };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter"};
  public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold",
content: "Bold", isToggle: true };
  public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
italic", content: "Italic", isToggle: true };
  public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
underline", content: "Underline", isToggle: true };

```

```

    public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontStyle, index: 3 };
    public fontSizeSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontSize, index: 3 };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group orientation="Column">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group orientation="Row">
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontStyleSettings">
          </e-ribbon-item>
          <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontSizeSettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collection>
  </e-ribbon-groups>

```



```

        <e-ribbon-items>
            <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
                </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
                </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
                </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

When the orientation is set to **Row** a group may have a maximum of three collections each of which may contain any number of items. When the orientation is set to **Column** a group may have any number of collections, each of which may contain one large-sized item or three medium/small-sized items. If two large-sized items are specified, it automatically converts into two medium/small-sized items.

Defining group header

You can use the [header](#) property to set the name for each group header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel } from
 '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
  "Merge format" }, { text: "Keep text only" }];
  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
  paste", content: "Paste", items: this.pasteOptions };
  public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold",
  content: "Bold", isToggle: true }
  public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
  italic", content: "Italic", isToggle: true }
  public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
  underline", content: "Underline", isToggle: true };
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
      <e-ribbon-group header="Font">
        <e-ribbon-collections>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Defining group icon

You can use the [groupIconCss](#) property to customize the icons in the group overflow button. When the ribbon size is adjusted, the group popup will appear.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
```

```

import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
imports: [ RibbonModule],
standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste", items: this.pasteOptions };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold",
content: "Bold", isToggle: true };
  public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
italic", content: "Italic", isToggle: true };
  public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
underline", content: "Underline", isToggle: true };
  public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontStyle, index: 3 };
  public fontSizeSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontSize, index: 3 };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>

```

```

        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
                    </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                        </e-ribbon-item>
                    <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                        </e-ribbon-item>
                    <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
                        </e-ribbon-item>
                    </e-ribbon-items>
                </e-ribbon-collection>
            </e-ribbon-collections>
        </e-ribbon-group>
        <e-ribbon-group header="Font" groupIconCss="e-icons e-bold">
            <e-ribbon-collections>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontStyleSettings">
                            </e-ribbon-item>
                        <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontSizeSettings">
                            </e-ribbon-item>
                        </e-ribbon-items>
                    </e-ribbon-collection>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
                            </e-ribbon-item>
                        <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
                            </e-ribbon-item>
                        <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
                            </e-ribbon-item>
                        </e-ribbon-items>
                    </e-ribbon-collection>
                </e-ribbon-collections>
            </e-ribbon-group>
        </e-ribbon-groups>
    </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Enabling group launcher icon

You can use the [showLauncherIcon](#) property to enable or disable the launcher icon for each group. By default, the property is set to `false`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" }
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
  content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
  format-painter", content: "Format Painter"};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
        paste" showLauncherIcon=true>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</e-ribbon>
```

```

        </e-ribbon-group>
    </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Customize launcher icon

You can use the [launcherIconCss](#) property to customize the launcher icon by applying the custom styles.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" }
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
  content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
  format-painter", content: "Format Painter"};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" launcherIconCss="e-icons e-description">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
        paste" showLauncherIcon=true>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                </e-ribbon-item>

```

```

        <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Defining group collapsible state

You can use the [isCollapsible](#) property to determine whether the group is collapsed or not during resize. By default, the property is set to `true`. To prevent the group from being collapsed, set the property to `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from "@syncfusion/ej2-angular-splitbuttons";
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
  public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste", items: this.pasteOptions };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter"};
  public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold",
content: "Bold", isToggle: true };
  public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
italic", content: "Italic", isToggle: true };
  public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
underline", content: "Underline", isToggle: true };

```

```

    public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontStyle, index: 3 };
    public fontSizeSettings: RibbonComboBoxSettingsModel = { dataSource:
this.fontSize, index: 3 };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Font" [isCollapsible]=false>
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontStyleSettings">
          </e-ribbon-item>
          <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontSizeSettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-groups>
</e-ribbon-tabs>

```



```

        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
                    </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Defining priority order for group collapse or expand

You can use the [priority](#) property to set the priority order for each group which should be collapsed or expanded on resizing. When collapsing, higher priority values are fetched first. When expanding, lower priority values are fetched first.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, RibbonComboBoxSettingsModel,
RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
imports: [ RibbonModule],
standalone: true,
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
    public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge format" }, { text: "Keep text only" }];
    public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
"20", "22", "24", "26", "28", "36", "48", "72", "96"];
    public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
"Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
"Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
"Windings"];
    public findOptions: ItemModel[] = [{ text: "Find", iconCss: "e-icons e-
search" }, { text: "Advanced find", iconCss: "e-icons e-search" }, { text:
"Go to", iconCss: "e-icons e-arrow-right" }];
    public selectOptions: ItemModel[] = [{ text: "Select All" }, { text:
"Select Objects" }];

```

```

    public findSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-search", content: "Find", items: this.findOptions };
    public replaceButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-replace", content: "Replace" };
    public selectSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-mouse-pointer", content: "Select", items: this.selectOptions };
    public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste", items: this.pasteOptions };
    public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
    public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
    public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
    public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold", content: "Bold", isToggle: true };
    public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-italic", content: "Italic", isToggle: true };
    public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-underline", content: "Underline", isToggle: true };
    public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3 };
    public fontSizeSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontSize, index: 3 };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-paste" priority=2>
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

```

        </e-ribbon-item>
        <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
<e-ribbon-group header="Font" groupIconCss="e-icons e-bold"
priority=0>
    <e-ribbon-collections>
        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontStyleSettings">
                    </e-ribbon-item>
                <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontSizeSettings">
                    </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
                    </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
<e-ribbon-group header="Editing" groupIconCss="e-icons e-
edit" priority=1>
    <e-ribbon-collections>
        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="findSettings">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="replaceButton">
                    </e-ribbon-item>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="selectSettings">
                    </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```
</ejs-ribbon>
```

Simplified layout

In simplified layout, the Ribbon component organizes the items and groups into a single row by setting the [activeLayout](#) property to [Simplified](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, RibbonDropDownSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions };
  public Simplified: DisplayMode = DisplayMode.Simplified;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" activelayout="Simplified">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
```

```

        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collection>
    <e-ribbon-items>
      <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
        </e-ribbon-item>
      <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
        </e-ribbon-item>
    </e-ribbon-items>
  </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
<e-ribbon-tab header="Insert">
  <e-ribbon-groups>
    <e-ribbon-group header="Tables">
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
              </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Enabling group overflow popup

You can use the [enableGroupOverflow](#) property to add a separate popup for the overflow items in the group while resizing. The overflow items will appear in the common overflow popup, located at the right end of the tab content if it is set to `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, RibbonComboBoxSettingsModel,
RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {

```

```

    public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
    "Merge format" }, { text: "Keep text only" }];
    public fontSize: string[] = ["8", "9", "10", "11", "12", "14", "16", "18",
    "20", "22", "24", "26", "28", "36", "48", "72", "96"];
    public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria",
    "Cambria Math", "Courier New", "Candara", "Georgia", "Impact", "Segoe Print",
    "Segoe Script", "Segoe UI", "Symbol", "Times New Roman", "Verdana",
    "Windings"];
    public findOptions: ItemModel[] = [{ text: "Find", iconCss: "e-icons e-
    search" }, { text: "Advanced find", iconCss: "e-icons e-search" }, { text:
    "Go to", iconCss: "e-icons e-arrow-right" }];
    public selectOptions: ItemModel[] = [{ text: "Select All" }, { text:
    "Select Objects" }];
    public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-
    paste", content: "Paste", items: this.pasteOptions };
    public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
    content: "Cut" };
    public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
    content: "Copy" };
    public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
    format-painter", content: "Format Painter" };
    public boldButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bold",
    content: "Bold", isToggle: true };
    public italicButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
    italic", content: "Italic", isToggle: true };
    public underlineButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
    underline", content: "Underline", isToggle: true };
    public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource:
    this.fontStyle, index: 3 };
    public fontSizeSettings: RibbonComboBoxSettingsModel = { dataSource:
    this.fontSize, index: 3 };
    public selectSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons
    e-mouse-pointer", content: "Select", items: this.selectOptions };
    public replaceButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
    replace", content: "Replace", };
    public findSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons
    e-search", content: "Find", items: this.findOptions }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
        paste">
          <e-ribbon-collections>
            <e-ribbon-collection>

```

```

        <e-ribbon-items>
            <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton">
                </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
        <e-ribbon-collection>
            <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                    </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
                    </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
    <e-ribbon-group header="Font" orientation=Row
groupIconCss="e-icons e-bold" enableGroupOverflow=true>
        <e-ribbon-collections>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontStyleSettings">
                        </e-ribbon-item>
                    <e-ribbon-item type="ComboBox"
[comboBoxSettings]="fontSizeSettings">
                        </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="Button"
[buttonSettings]="boldButton">
                        </e-ribbon-item>
                    <e-ribbon-item type="Button"
[buttonSettings]="italicButton">
                        </e-ribbon-item>
                    <e-ribbon-item type="Button"
[buttonSettings]="underlineButton">
                        </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
        </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Editing" groupIconCss="e-icons e-
edit">
        <e-ribbon-collections>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="SplitButton"
[splitButtonSettings]="findSettings">
                        </e-ribbon-item>

```

```

        <e-ribbon-item type="Button"
[buttonSettings]="replaceButton">
        </e-ribbon-item>
        <e-ribbon-item type="SplitButton"
[splitButtonSettings]="selectSettings">
        </e-ribbon-item>
    </e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Minimized state

You can hide the Ribbon contents and display only the tab headers by double-clicking on the tab header. In minimized state, the Ribbon component expands to its normal state when click on the tab header.

You can use the [isMinimized](#) property to change the Ribbon component to minimized state. By default, the value is `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel,RibbonDropDownSettingsModel } from
'@syncfusion/ej2-angular-ribbon';
import { ItemModel } from "@syncfusion/ej2-angular-splitbuttons";
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This
device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-
table", content: "Table", items: this.tableOptions };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```



```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" [isMinimized]="true">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
<e-ribbon-tab header="Insert">
  <e-ribbon-groups>
    <e-ribbon-group header="Tables">
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="DropDown"
[dropDownSettings]="tableSettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Show or hide the layout switcher

You can use the [hideLayoutSwitcher](#) property to show/hide the Ribbon layout switcher button. By default, the value is `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon';
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons';
import { Component, ViewChild } from '@angular/core';
import { RibbonButtonSettingsModel, RibbonDropDownSettingsModel, DisplayMode,
RibbonComponent } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from "@syncfusion/ej2-angular-splitbuttons";
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [ RibbonModule, CheckBoxModule ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  @ViewChild('ribbon') ribbon: RibbonComponent;

  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions };
  public Simplified: DisplayMode = DisplayMode.Simplified;
  public onChange(args: ChangeEventArgs) {
    this.ribbon.hideLayoutSwitcher = !args.checked;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-checkbox label="Show/Hide Layout Switcher" [checked]="true"
(change)='onChange($event)'></ejs-checkbox>
<ejs-ribbon id="ribbon" #ribbon [hideLayoutSwitcher]="false">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button" [buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-collection>
          </e-ribbon-collections>
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

```

        </e-ribbon-items>
    </e-ribbon-collection>
    <e-ribbon-collection>
        <e-ribbon-items>
            <e-ribbon-item type="Button" [buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button" [buttonSettings]="copyButton">
            </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
<e-ribbon-tab header="Insert">
    <e-ribbon-groups>
        <e-ribbon-group header="Tables">
            <e-ribbon-collections>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item
                            type="DropDown"
                            [dropDownSettings]="tableSettings"
                        >
                        </e-ribbon-item>
                    </e-ribbon-items>
                </e-ribbon-collection>
            </e-ribbon-collections>
        </e-ribbon-group>
    </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

File Menu

The Ribbon component provides a built-in file menu that allows you to add menu items for performing specific actions. The file menu can be enabled by setting the [fileMenu](#) property.

Visibility

You can show the file menu by setting the [visible](#) property to `true`. By default, the file menu is hidden.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from
'syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {

```

```

    public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
    public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
    public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
    public fileSettings: FileMenuSettingsModel = {
        menuItems: [{ text: "New", iconCss: "e-icons e-file-new", id: "new" }],
        visible: true
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                  </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>

```

Adding menu items

The menu items can be added to the file menu using the [menuItems](#) property as an array of [MenuItemModel](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from
 '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-
file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  public fileSettings: FileMenuSettingsModel = {
    menuItems: this.fileOptions,
    visible: true
  };
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [fileMenu]="fileSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collection>
          </e-ribbon-collections>
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

```

        <e-ribbon-items>
            <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Open submenu on click

You can open the submenu on menu item click, by setting the [showItemOnClick](#) property to `true`. By default, the submenu will open on mouse hover.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from
'syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from '@syncfusion/ej2-navigations';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public fileOptions: MenuItemModel[] =[
    { text: 'New', iconCss:'e-icons e-file-new', id: 'new' },
    { text: 'Open', iconCss:'e-icons e-folder-open', id: 'open' },
    { text: 'Rename', iconCss:'e-icons e-rename', id: 'rename' },
    {
      text: 'Save as',
      iconCss:'e-icons e-save',
      id: 'save',
      items: [
        { text: 'Microsoft Word (.docx)' },
        { text: 'Microsoft Word 97-2003(.doc)' },
        { text: 'Download as PDF' }
      ]
    }
  ]
}

```

```

];
public fileSettings: FileMenuSettingsModel = {
  menuItems: this.fileOptions,
  showItemOnClick: true,
  visible: true
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Custom header text

You can define the file menu header text content by using the [text](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';

```

```
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from
'@syncfusion/ej2-angular-ribbon';
@Component({
imports: [ RibbonModule ],
providers: [ RibbonFileMenuService ],
standalone: true,
selector: 'app-root',
templateUrl: './app.component.html'
})
export class AppComponent {
public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
public fileSettings: FileMenuSettingsModel = {
text: 'App',
menuItems: [{ text: "New", iconCss: "e-icons e-file-new", id: "new" }],
visible: true
};
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
```



```

        </e-ribbon-group>
    </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Ribbon Backstage

The Ribbon supports backstage view which is an addition to the traditional file menu. It displays information like application settings, user details, etc. The backstage view can be enabled by setting the [backStageMenu](#) property.

The backstage view options are displayed on the left, while the content of each option is shown on the right.

Adding backstage items

The menu items can be added to the backstage view by using the [items](#) property. You can show the backstage view by setting the [visible](#) property to `true`. By default, the backstage view is hidden.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, BackStageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "home":
      {
        content = "<div id='home-wrapper' style='padding: 20px;'><div id='new-section' class='new-wrapper'><div class='section-title'> New
        </div><div class='category_container'><div class='doc_category_image'></div><span class='doc_category_text'> New
        document </span></div></div><div id='block-wrapper'><div class='section-
        title'> Recent </div><div class='section-content' style='padding: 12px 0px;
        cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
        link'></span> </td><td> <span style='display: block; font-size: 14px'>
        Ribbon.docx </span><span style='font-size: 12px'> EJ2 >> Components >>

```

```

Navigations >> Ribbon >> default
</span></td></tr></tbody></table></div></div></div>";
        break;
    }
    case "new":
    {
        content = "<div id='new-section' class='new-wrapper'><div
class='section-title'> New </div><div class='category_container'><div
class='doc_category_image'></div> <span class='doc_category_text'> New
document </span></div></div>";
        break;
    }
    case "open":
    {
        content = "<div id='open-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'> Open
in Desktop App </span><span style='font-size: 12px'> Use the full
functionality of Ribbon </span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-
sheet'></span> </td><td> <span style='display: block; font-size: 14px'>
Protect Document </span><span style='font-size: 12px'>To prevent accidental
changes, this document has been set to open as view-
only.</span></td></tr></tbody></table></div></div>";
        break;
    }
    }
    return content;
}

public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
this.getBackstageContent('home') },
    { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
this.getBackstageContent('new') },
    { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open', content:
this.getBackstageContent('open') }
];
public backstageSettings: BackStageMenuModel = {
    text: 'File',
    visible: true,
    items: this.menuItems,
    backButton: {
        text: 'Close',
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Adding footer items

You can use the [isFooter](#) property in the [items](#) collection to add the backstage view footer items. By default, the value is false.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, BackStageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonBackstageService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };

```

```

public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
        case "home":
        {
            content = "<div id='home-wrapper' style='padding: 20px;'><div
id='new-section' class='new-wrapper'><div class='section-title'> New
</div><div class='category_container'><div
class='doc_category_image'></div><span class='doc_category_text'> New
document </span></div></div><div id='block-wrapper'><div class='section-
title'> Recent </div><div class='section-content' style='padding: 12px 0px;
cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'>
Ribbon.docx </span><span style='font-size: 12px'> EJ2 >> Components >>
Navigations >> Ribbon >> default
</span></td></tr></tbody></table></div></div></div>";
            break;
        }
        case "new":
        {
            content = "<div id='new-section' class='new-wrapper'><div
class='section-title'> New </div><div class='category_container'><div
class='doc_category_image'></div> <span class='doc_category_text'> New
document </span></div></div>";
            break;
        }
        case "open":
        {
            content = "<div id='open-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'> Open
in Desktop App </span><span style='font-size: 12px'> Use the full
functionality of Ribbon </span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-
sheet'></span> </td><td> <span style='display: block; font-size: 14px'>
Protect Document </span><span style='font-size: 12px'>To prevent accidental
changes, this document has been set to open as view-
only.</span></td></tr></tbody></table></div></div>";
            break;
        }
        case "account":
        {
            content = "<div id='account-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-
people'></span> </td><td> <span style='display: block; font-size:
14px'>Account type</span><span style='font-size:
12px'>Administrator</span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-
password'></span> </td><td> <span style='display: block; font-size:
14px'>Password protected</span><span style='font-size:
12px'>Yes</span></td></tr></tbody></table></div></div>";
        }
    }
}

```

```

        break;
    }
}
return content;
}
public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
this.getBackstageContent('home') },
    { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
this.getBackstageContent('new') },
    { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open', content:
this.getBackstageContent('open') },
    { separator: true, isFooter: true },
    { id: 'account', text: 'Account', isFooter: true, content:
this.getBackstageContent('account') }
];
public backstageSettings: BackStageMenuModel = {
    text: 'File',
    visible: true,
    items: this.menuItems,
    backButton: {
        text: 'Close',
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

```

        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Adding separator

The separators are horizontal lines used to separate the backstage view items. You can use the [separator](#) property to split the menu items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, BackStageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonBackstageService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "home":
      {
        content = "<div id='home-wrapper' style='padding: 20px;'><div id='new-section' class='new-wrapper'><div class='section-title'> New</div><div class='category_container'><div class='doc_category_image'></div><span class='doc_category_text'> New document</span></div></div><div id='block-wrapper'><div class='section-title'> Recent</div><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-link'></span> </td><td> <span style='display: block; font-size: 14px'> Ribbon.docx</span><span style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >> default</span></td></tr></tbody></table></div></div></div>";
        break;
      }
      case "new":
      {

```

```

        content = "<div id='new-section' class='new-wrapper'><div
class='section-title'> New </div><div class='category_container'><div
class='doc_category_image'></div> <span class='doc_category_text'> New
document </span></div></div>";
        break;
    }
    case "open":
    {
        content = "<div id='open-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'> Open
in Desktop App </span><span style='font-size: 12px'> Use the full
functionality of Ribbon </span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-
sheet'></span> </td><td> <span style='display: block; font-size: 14px'>
Protect Document </span><span style='font-size: 12px'>To prevent accidental
changes, this document has been set to open as view-
only.</span></td></tr></tbody></table></div></div>";
        break;
    }
    case "print":
    {
        content = "<div id='print-content' style='min-width: 300px;
padding: 20px;'><h2>Print this document</h2><button id='togglebtn' class='e-
control e-btn e-lib e-flat e-primary'><span class='e-btn-icon e-btn-sb-icons
e-icons e-print e-icon-left'></span>Print</button></div>";
        break;
    }
    }
    return content;
}

public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
this.getBackstageContent('home') },
    { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
this.getBackstageContent('new') },
    { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open', content:
this.getBackstageContent('open') },
    { separator: true },
    { id: 'print', text: 'Print', content: this.getBackstageContent('print')
}
];

public backstageSettings: BackStageMenuModel = {
    text: 'File',
    visible: true,
    items: this.menuItems,
    backButton: {
        text: 'Close',
    }
}
}
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Back button

You can use the [backButton](#) property to customize the text and icon of the close button using the [text](#) and [iconCss](#) property. You can show the back button by setting the [visible](#) property to `true`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, BackStageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonBackstageService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```



```

})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public homeContentTemplate() {
    return "<div id='home-wrapper' style='padding: 20px;'><div id='new-section' class='new-wrapper'><div class='section-title'> New </div><div class='category_container'><div class='doc_category_image'></div><span class='doc_category_text'> New document </span></div></div><div id='block-wrapper'><div class='section-title'> Recent </div><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td><span class='doc_icon e-icons e-open-link'></span> </td><td> <span style='display: block; font-size: 14px'> Ribbon.docx </span><span style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >> default </span></td></tr></tbody></table></div></div></div>";
  }
  public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content: this.homeContentTemplate() }
  ];
  public backstageSettings: BackStageMenuModel = {
    visible: true,
    items: this.menuItems,
    backButton: {
      text: 'Close',
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

```

        <e-ribbon-items>
            <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Backstage target

The [target](#) property specifies the element selector in which backstage will be displayed. The target element should have the position as relative, else the backstage will be positioned nearest to the relative element. By default, the backstage is positioned to ribbon element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, BackstageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "home":
        {
          content = "<div id='home-wrapper' style='padding: 20px;'><div
id='new-section' class='new-wrapper'><div class='section-title'> New
</div><div class='category_container'><div
class='doc_category_image'></div><span class='doc_category_text'> New
document </span></div></div><div id='block-wrapper'><div class='section-
title'> Recent </div><div class='section-content' style='padding: 12px 0px;
cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'>

```

```

Ribbon.docx </span><span style='font-size: 12px'> EJ2 >> Components >>
Navigations >> Ribbon >> default
</span></td></tr></tbody></table></div></div></div>";
        break;
    }
    case "new":
    {
        content = "<div id='new-section' class='new-wrapper'><div
class='section-title'> New </div><div class='category_container'><div
class='doc_category_image'></div> <span class='doc_category_text'> New
document </span></div></div>";
        break;
    }
    case "open":
    {
        content = "<div id='open-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'> Open
in Desktop App </span><span style='font-size: 12px'> Use the full
functionality of Ribbon </span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-
sheet'></span> </td><td> <span style='display: block; font-size: 14px'>
Protect Document </span><span style='font-size: 12px'>To prevent accidental
changes, this document has been set to open as view-
only.</span></td></tr></tbody></table></div></div>";
        break;
    }
    }
    return content;
}
}
public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
this.getBackstageContent('home') },
    { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
this.getBackstageContent('new') },
    { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open', content:
this.getBackstageContent('open') }
];
public backstageSettings: BackStageMenuModel = {
    text: 'File',
    visible: true,
    items: this.menuItems,
    target: '#targetElement',
    backButton: {
        text: 'Close',
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<div id="targetElement"></div>
<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
                </e-ribbon-item>
              </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Template

You can use the [template](#) property to modify the backstage view menu items and their contents.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewChild } from '@angular/core';
import { RibbonButtonSettingsModel, BackStageMenuModel } from
'@syncfusion/ej2-angular-ribbon';
import { Ribbon } from '@syncfusion/ej2-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonBackstageService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  @ViewChild('ribbon')
```

```

public ribbonObj: Ribbon;
public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
public contentClick(id: string) {
    let ribbonEle = document.getElementById('ribbon');
    if(ribbonEle) {
        let content = ribbonEle.querySelector('.content-open');
        if (content) {
            content.classList.replace('content-open', 'content-close');
        }
        ribbonEle.querySelector('#' + id + '-wrapper')?.classList.add('content-
open');
    }
}
public closeContent() {
    if (this.ribbonObj) {
        (this.ribbonObj.element.querySelector('#ribbon_backstagepopup') as
HTMLElement).style.display = 'none';
    }
}
public ribbonCreated() {
    if (this.ribbonObj) {
        this.ribbonObj.element.querySelector('.e-ribbon-
backstage').addEventListener('click', this.displayPopup);
    }
}
public displayPopup() {
    let backstagePopup =
this.ribbonObj.element.querySelector('#ribbon_backstagepopup') as
HTMLElement;
    if (backstagePopup) {
        backstagePopup.style.display = 'block';
    }
}
public homeContentTemplate() {
    return "<div id='temp-content' style='width: 550px; height: 350px;
display: flex'><div id='items-wrapper' style='width: 130px; height:100%;
background: #779de8;'><ul><li id='close'
(click)='this.closeContent(this.id)'><span class='e-icons e-
close'></span>Close</li><li id='new'
(click)='this.contentClick(this.id)'><span class='e-icons e-file-
new'></span>New</li><li id='open' (click)='this.contentClick(this.id)'><span
class='e-icons e-folder-open'></span>Open</li><li id='save'
(click)='this.contentClick(this.id)'><span class='e-icons e-
save'></span>Save</li></ul></div><div id='content-wrapper'><div id='new-
wrapper' class='content-open' style='padding: 20px;'><div id='new-section'
class='new-wrapper'><div class='section-title'>New</div><div
class='category_container'><div class='doc_category_image'></div><span
class='doc_category_text'>New document</span></div></div><div id='save-
wrapper' class='content-close' style='padding: 20px;'><div class='section-
content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td><span class='doc_icon e-icons e-
save'></span></td><td><span style='display: block; font-size: 14px'>Save

```

```

as</span><span style='font-size: 12px'>Save as copy
online</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px cursor: pointer'><table><tbody><tr><td><span
class='doc_icon e-icons e-rename'></span></td><td><span style='display:
block; font-size: 14px'>Rename</span><span style='font-size: 12px'>Rename
this file.</span></td></tr></tbody></table></div></div><div id='open-wrapper'
class='content-close' style='padding: 20px;'><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td><span
class='doc_icon e-icons e-open-link'></span></td><td><span style='display:
block; font-size: 14px'>Ribbon.docx</span><span style='font-size: 12px'>EJ2
>> Components >> Navigations >> Ribbon >>
default</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td><span
class='doc_icon e-icons e-open-link'></span></td><td><span style='display:
block; font-size: 14px'>Classic_layout.docx</span><span style='font-size:
12px'>EJ2 >> Components >> Navigations >> Ribbon >>
layouts</span></td></tr></tbody></table></div><div class='section-content'
style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td><span
class='doc_icon e-icons e-open-link'></span></td><td><span style='display:
block; font-size: 14px'>Simplified_layout.docx</span><span style='font-size:
12px'>EJ2 >> Components >> Navigations >> Ribbon >>
layouts</span></td></tr></tbody></table></div></div></div></div>";
    }
    public backstageSettings: BackStageMenuModel = {
        visible: true,
        template: this.homeContentTemplate()
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings"
(created)="ribbonCreated()">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
                </e-ribbon-item>
              </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">

```

```

        </e-ribbon-item>
        <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Setting width and height

You can customize the height and width of the backstage view using the [height](#) and [width](#) property. By default, dimensions are set based on the content added.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel, BackstageMenuModel, BackstageItemModel }
from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
paste", content: "Paste" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "home":
        {
          content = "<div id='home-wrapper' style='padding: 20px;'><div
id='new-section' class='new-wrapper'><div class='section-title'> New
</div><div class='category_container'><div
class='doc_category_image'></div><span class='doc_category_text'> New
document </span></div></div><div id='block-wrapper'><div class='section-
title'> Recent </div><div class='section-content' style='padding: 12px 0px;
cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'>
Ribbon.docx </span><span style='font-size: 12px'> EJ2 >> Components >>
Navigations >> Ribbon >> default
</span></td></tr></tbody></table></div></div></div>";
          break;

```

```

    }
    case "new":
    {
        content = "<div id='new-section' class='new-wrapper'><div
class='section-title'> New </div><div class='category_container'><div
class='doc_category_image'></div> <span class='doc_category_text'> New
document </span></div></div>";
        break;
    }
    case "open":
    {
        content = "<div id='open-content' style='padding: 20px;'><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-
link'></span> </td><td> <span style='display: block; font-size: 14px'> Open
in Desktop App </span><span style='font-size: 12px'> Use the full
functionality of Ribbon </span></td></tr></tbody></table></div><div
class='section-content' style='padding: 12px 0px; cursor:
pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-
sheet'></span> </td><td> <span style='display: block; font-size: 14px'>
Protect Document </span><span style='font-size: 12px'>To prevent accidental
changes, this document has been set to open as view-
only.</span></td></tr></tbody></table></div></div>";
        break;
    }
    }
    return content;
}
}
public menuItems: BackstageItemModel[] = [
    { id: 'home', text: 'Home', iconCss: 'e-icons e-home', content:
this.getBackstageContent('home') },
    { id: 'new', text: 'New', iconCss: 'e-icons e-file-new', content:
this.getBackstageContent('new') },
    { id: 'open', text: 'Open', iconCss: 'e-icons e-folder-open', content:
this.getBackstageContent('open') }
];
public backstageSettings: BackStageMenuModel = {
    height: "350px",
    width: "500px",
    text: 'File',
    visible: true,
    items: this.menuItems,
    backButton: {
        text: 'Close',
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```


APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="pasteButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

[Adding Backstage events](#)[Ribbon Contextual Tabs](#)

The Ribbon Contextual tabs are similar to the Ribbon tabs that are displayed on demand based on their needs, such as an image or a table tabs. It supports adding all built-in and custom ribbon items to perform specific actions.

[Visible tabs](#)

You can utilize the [visible](#) property within each `e-ribbon-contextual-tag` directive to control the visibility of each contextual tab.

[Adding contextual tabs](#)

You can utilize the `e-ribbon-contextual-tabs` tag directive to add contextual tabs in the Ribbon where you can add multiple tabs based on your needs.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonContextualTabService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';

```

```
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonContextualTabService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
  content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Format Painter" };
  public headerButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-text-header",
  content: "Text Header" };
  public wrapButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-text-wrap",
  content: "Text Wrap" };
  public annotationButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-text-annotation",
  content: "Text Annotation" };
  public altButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-text-alternative",
  content: "Alt Text" };
  public forwardButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bring-forward",
  content: "Bring Forward" };
  public backwardButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-send-backward",
  content: "Send Backward" };
  public selectionButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-show-hide-panel",
  content: "Selection Pane" };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-contextual-tabs>
    <e-ribbon-contextual-tab [visible]="true">
      <e-ribbon-tabs>
        <e-ribbon-tab header="Shape Format">
          <e-ribbon-groups>
            <e-ribbon-group header="Text decoration"
[showLauncherIcon]="true">
              <e-ribbon-collections>
                <e-ribbon-collection>
                  <e-ribbon-items>
                    <e-ribbon-item type="Button"
[buttonSettings]="headerButton"> </e-ribbon-item>
```

```

                                <e-ribbon-item type="Button"
[buttonSettings]="wrapButton"> </e-ribbon-item>
                                <e-ribbon-item type="Button"
[buttonSettings]="annotationButton"> </e-ribbon-item>
                                </e-ribbon-items>
                                </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                <e-ribbon-group header="Accessibility">
                                <e-ribbon-collections>
                                    <e-ribbon-collection>
                                        <e-ribbon-items>
                                            <e-ribbon-item type="Button"
[buttonSettings]="altButton"> </e-ribbon-item>
                                        </e-ribbon-items>
                                    </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                <e-ribbon-group header="Arrange"
[showLauncherIcon]="true">
                                    <e-ribbon-collections>
                                        <e-ribbon-collection>
                                            <e-ribbon-items>
                                                <e-ribbon-item type="Button"
[buttonSettings]="forwardButton"> </e-ribbon-item>
                                                <e-ribbon-item type="Button"
[buttonSettings]="backwardButton"> </e-ribbon-item>
                                                <e-ribbon-item type="Button"
[buttonSettings]="selectionButton"> </e-ribbon-item>
                                            </e-ribbon-items>
                                        </e-ribbon-collection>
                                    </e-ribbon-collections>
                                </e-ribbon-group>
                                </e-ribbon-groups>
                                </e-ribbon-tab>
                                </e-ribbon-tabs>
                                </e-ribbon-contextual-tab>
                                </e-ribbon-contextual-tabs>
                                <e-ribbon-tabs>
                                    <e-ribbon-tab header="Home">
                                        <e-ribbon-groups>
                                            <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
                                                <e-ribbon-collections>
                                                    <e-ribbon-collection>
                                                        <e-ribbon-items>
                                                            <e-ribbon-item type="Button"
[buttonSettings]="cutButton"> </e-ribbon-item>
                                                            <e-ribbon-item type="Button"
[buttonSettings]="copyButton"> </e-ribbon-item>
                                                            <e-ribbon-item type="Button"
[buttonSettings]="formatButton"> </e-ribbon-item>
                                                        </e-ribbon-items>
                                                    </e-ribbon-collection>
                                                </e-ribbon-collections>
                                            </e-ribbon-group>
                                        </e-ribbon-groups>

```

```

    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

Selected tabs

By using the [isSelected](#) property, you can control the selected state of each contextual tab and indicates which tab is currently active.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonContextualTabService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonContextualTabService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
  content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Format Painter" };
  public styleButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-style", content: "Style" };
  public textButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-font-name", content: "Text Box" };
  public paintButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paint-bucket", content: "Paint" };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-contextual-tabs>
    <e-ribbon-contextual-tab [isSelected]="true" [visible]="true">
      <e-ribbon-tabs>
        <e-ribbon-tab header="Styles">
          <e-ribbon-groups>

```

```

        <e-ribbon-group header="Style"
[showLauncherIcon]="true">
            <e-ribbon-collections>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item type="Button"
[buttonSettings]="styleButton"> </e-ribbon-item>
                        <e-ribbon-item type="Button"
[buttonSettings]="textButton"> </e-ribbon-item>
                        <e-ribbon-item type="Button"
[buttonSettings]="paintButton"> </e-ribbon-item>
                    </e-ribbon-items>
                </e-ribbon-collection>
            </e-ribbon-collections>
        </e-ribbon-group>
    </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</e-ribbon-contextual-tab>
</e-ribbon-contextual-tabs>
<e-ribbon-tabs>
    <e-ribbon-tab header="Home">
        <e-ribbon-groups>
            <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
                <e-ribbon-collections>
                    <e-ribbon-collection>
                        <e-ribbon-items>
                            <e-ribbon-item type="Button"
[buttonSettings]="cutButton"> </e-ribbon-item>
                            <e-ribbon-item type="Button"
[buttonSettings]="copyButton"> </e-ribbon-item>
                            <e-ribbon-item type="Button"
[buttonSettings]="formatButton"> </e-ribbon-item>
                        </e-ribbon-items>
                    </e-ribbon-collection>
                </e-ribbon-collections>
            </e-ribbon-group>
        </e-ribbon-groups>
    </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Methods

Show tab

You can use the [showTab](#) method to make the specific Contextual tab visible in the Ribbon.

Hide tab

You can use the [hideTab](#) method to hide specific Contextual tab in the Ribbon.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonContextualTabService } from '@syncfusion/ej2-
angular-ribbon'

```

```
import { Component, ViewChild } from "@angular/core";
import { RibbonButtonSettingsModel, DisplayMode } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonContextualTabService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;
  @ViewChild('showContextual') showElement!: HTMLElement;
  @ViewChild('hideContextual') hideElement!: HTMLElement;
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
  content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Format Painter" };
  public forwardButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-bring-forward", content: "Bring Forward" };
  public backwardButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-send-backward", content: "Send Backward" };
  public selectionButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-show-hide-panel", content: "Selection Pane" };
  toggleRibbonTabs = (value: string) => {
    (this.ribbon as any)[value]('ArrangeView', true);
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<button (click)="toggleRibbonTabs('showTab')" class="e-btn"> Show tab
</button>
<button (click)="toggleRibbonTabs('hideTab')" class="e-btn"> Hide tab
</button>
<ejs-ribbon #ribbon id="ribbon" style="margin-top: 30px;">
  <e-ribbon-contextual-tabs>
    <e-ribbon-contextual-tab>
      <e-ribbon-tabs>
        <e-ribbon-tab id="ArrangeView" header="Arrange & View">
          <e-ribbon-groups>
            <e-ribbon-group header="Arrange"
[showLauncherIcon]="true">
              <e-ribbon-collections>
                <e-ribbon-collection>
                  <e-ribbon-items>
```

```

                                <e-ribbon-item type="Button"
[buttonSettings]="forwardButton"> </e-ribbon-item>
                                <e-ribbon-item type="Button"
[buttonSettings]="backwardButton"> </e-ribbon-item>
                                <e-ribbon-item type="Button"
[buttonSettings]="selectionButton"> </e-ribbon-item>
                                </e-ribbon-items>
                                </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                </e-ribbon-groups>
                                </e-ribbon-tab>
                                </e-ribbon-tabs>
                                </e-ribbon-contextual-tab>
                                </e-ribbon-contextual-tabs>
                                <e-ribbon-tabs>
                                <e-ribbon-tab header="Home">
                                <e-ribbon-groups>
                                <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
                                <e-ribbon-collections>
                                <e-ribbon-collection>
                                <e-ribbon-items>
                                <e-ribbon-item type="Button"
[buttonSettings]="cutButton"> </e-ribbon-item>
                                <e-ribbon-item type="Button"
[buttonSettings]="copyButton"> </e-ribbon-item>
                                <e-ribbon-item type="Button"
[buttonSettings]="formatButton"> </e-ribbon-item>
                                </e-ribbon-items>
                                </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                </e-ribbon-groups>
                                </e-ribbon-tab>
                                </e-ribbon-tabs>
                                </ejs-ribbon>

```

Ribbon Keytip

The Ribbon supports keyboard navigations to interact the ribbon items using the keytips which can be enabled by setting the [enableKeyTips](#) property.

The keytips will be shown when the **Alt + Windows/Command** keys are pressed.

Ribbon items keytip

You can add keytips to all the ribbon items by using the [keyTip](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGroupButtonService, RibbonGalleryService,
RibbonKeyTipService, RibbonColorPickerService } from '@syncfusion/ej2-
angular-ribbon'
import { Component, ViewChild } from "@angular/core";

```

```

import { RibbonItemSize, RibbonButtonSettingsModel,
RibbonGroupButtonSelection, RibbonSplitButtonSettingsModel, RibbonComponent,
RibbonComboBoxSettingsModel, RibbonGroupButtonSettingsModel,
RibbonColorPickerSettingsModel, RibbonGallerySettingsModel,
RibbonDropDownSettingsModel, RibbonCheckBoxSettingsModel } from
'@syncfusion/ej2-angular-ribbon';
@Component({
imports: [ RibbonModule],
providers: [ RibbonGroupButtonService, RibbonGalleryService,
RibbonKeyTipService, RibbonColorPickerService ],
standalone: true,
selector: 'app-root',
templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;

  public largeSize = RibbonItemSize.Large;
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons
e-paste", content: "Paste", items: [{ text: 'Keep Source Format' }, { text:
'Merge format' }, { text: 'Keep text only' }] };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public pasteOptions = { title: "Paste", cssClass: 'custom-tooltip',
content: "Paste content here.</br> Add content on the clipboard to your
document.", iconCss: "e-icons e-paste" }
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" }
  public styleOptions: RibbonComboBoxSettingsModel = {
    dataSource: ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math",
"Courier New", "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script",
"Segoe UI", "Symbol", "Times New Roman", "Verdana", "Windings"],
    index: 2,
    width: '150px',
    allowFiltering: true
  }
  public sizeOptions: RibbonComboBoxSettingsModel = {
    dataSource: ["8", "9", "10", "11", "12", "14", "16", "18", "20", "22",
"24", "26", "28", "36", "48", "72", "96"],
    index: 4,
    width: "65px"
  }
  public groupButtonSingle: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      { iconCss: 'e-icons e-bold', keyTip: '1', selected: true },
      { iconCss: 'e-icons e-italic', keyTip: '2' },
      { iconCss: 'e-icons e-underline', keyTip: '3' },
      { iconCss: 'e-icons e-strikethrough', keyTip: '4' },
      { iconCss: 'e-icons e-change-case', keyTip: '5' }
    ]
  }
  public colorPicker: RibbonColorPickerSettingsModel = { value: "#123456" }
  public gallerySettings: RibbonGallerySettingsModel = {
    itemCount: 3,

```



```

    groups: [{
      itemWidth: '100',
      header: 'Styles',
      items: [{
        content: 'Normal'
      }, {
        content: 'No Spacing'
      }, {
        content: 'Heading 1'
      }, {
        content: 'Heading 2'
      }, {
        content: 'Heading 3'
      }, {
        content: 'Heading 4'
      }, {
        content: 'Heading 5'
      }]
    }]
  }
  public tableSettings: RibbonDropDownSettingsModel = {
    iconCss: 'e-icons e-table', content: 'Table',
    items: [{ text: 'Insert Table' }, { text: 'Draw Table' }, { text:
'Convert Table' }, { text: 'Excel SpreadSheet' }]
  }
  public rulerSettings: RibbonCheckBoxSettingsModel = { label: "Ruler",
checked: false }
  public gridSettings: RibbonCheckBoxSettingsModel = { label: "Gridlines",
checked: false }
  public navigationSettings: RibbonCheckBoxSettingsModel = { label:
"Navigation Pane", checked: true }
  ribbonCreated(): void {
    this.ribbon.ribbonKeyTipModule.showKeyTips('H');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon #ribbon id="ribbon" (created)="ribbonCreated()"
[enableKeyTips]="true">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home" keyTip="H">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-paste"
keyTip="CD">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>

```

```

        <e-ribbon-item type="SplitButton"
[allowedSizes]="largeSize" keyTip="PA" [ribbonTooltipSettings]="pasteOptions"
[splitButtonSettings]="pasteSettings">
        </e-ribbon-item>
    </e-ribbon-items>
</e-ribbon-collection>
<e-ribbon-collection>
    <e-ribbon-items>
        <e-ribbon-item type="Button" keyTip="CU"
[buttonSettings]="cutButton">
        </e-ribbon-item>
        <e-ribbon-item type="Button" keyTip="CO"
[buttonSettings]="copyButton">
        </e-ribbon-item>
        <e-ribbon-item type="Button" keyTip="CS"
[buttonSettings]="formatButton">
        </e-ribbon-item>
    </e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
    <e-ribbon-group header="Font" orientation="Row"
[enableGroupOverflow]="true" keyTip="FB" groupIconCss="e-icons e-bold"
cssClass="font-group" >
        <e-ribbon-collections>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="ComboBox" keyTip="01"
[comboBoxSettings]="styleOptions" >
                    </e-ribbon-item>
                    <e-ribbon-item type="ComboBox" keyTip="02"
[comboBoxSettings]="sizeOptions">
                    </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="GroupButton" keyTip="GB"
[groupButtonSettings]="groupButtonSingle"></e-ribbon-item>
                    <e-ribbon-item type="ColorPicker" keyTip="CP"
[colorPickerSettings]="colorPicker" >
                    </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
        </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" groupIconCss="e-icons e-paste"
[showLauncherIcon]=true>
        <e-ribbon-collections>
            <e-ribbon-collection>
                <e-ribbon-items>
                    <e-ribbon-item type="Gallery" keyTip="GY"
[gallerySettings]="gallerySettings">
                    </e-ribbon-item>
                </e-ribbon-items>
            </e-ribbon-collection>
        </e-ribbon-collections>
    </e-ribbon-collections>

```

```

    </e-ribbon-group>
    <e-ribbon-group header="Tables" groupIconCss="e-icons e-table">
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="DropDown" keyTip="T"
[allowedSizes]="largeSize" [dropDownSettings]="tableSettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
  <e-ribbon-group header="show" groupIconCss="e-icons e-copy">
    <e-ribbon-collections>
      <e-ribbon-collection>
        <e-ribbon-items>
          <e-ribbon-item type="CheckBox" keyTip="R1"
[checkBoxSettings]="rulerSettings">
          </e-ribbon-item>
          <e-ribbon-item type="CheckBox" keyTip="R2"
[checkBoxSettings]="gridSettings">
          </e-ribbon-item>
          <e-ribbon-item type="CheckBox" keyTip="R3"
[checkBoxSettings]="navigationSettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

STYLES.CSS

```

@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';
.custom-tooltip.e-paste{
  font-size: 50px;
}

```

File menu keytip

You can add keytips to the file menu by using the [keyTip](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonFileMenuService, RibbonKeyTipService } from
 '@syncfusion/ej2-angular-ribbon'
import { Component, ViewChild } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel,
RibbonSplitButtonSettingsModel, RibbonComponent } from '@syncfusion/ej2-
angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonFileMenuService, RibbonKeyTipService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;

  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons
e-paste", content: "Paste", items: [{ text: 'Keep Source Format' }, { text:
'Merge format' }, { text: 'Keep text only' }] };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public fileMenuSettings: FileMenuSettingsModel = {
    keyTip: "F",
    visible: true,
    menuItems: [
      { text: "New", iconCss: "e-icons e-file-new", id: "new" },
      { text: "Open", iconCss: "e-icons e-folder-open", id: "open" },
      { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
      { text: "Save as", iconCss: "e-icons e-save", id: "save" }
    ]
  };
  ribbonCreated(): void {
    this.ribbon.ribbonKeyTipModule.showKeyTips();
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon #ribbon id="ribbon" (created)="ribbonCreated()"
[enableKeyTips]="true" [fileMenu]="fileMenuSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>

```

```

        <e-ribbon-items>
          <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

STYLES.CSS

```

@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';

```

Backstage menu keytip

You can add keytips to backstage menu items by using the [keyTip](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonBackstageService, RibbonKeyTipService } from
 '@syncfusion/ej2-angular-ribbon'
import { Component, ViewChild } from "@angular/core";
import { RibbonButtonSettingsModel, BackStageMenuItemModel, BackstageItemModel,
RibbonSplitButtonSettingsModel, RibbonComponent } from '@syncfusion/ej2-
angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonBackstageService, RibbonKeyTipService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})

```

```

export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;

  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste", items: [{ text: 'Keep Source Format' }, { text: 'Merge format' }, { text: 'Keep text only' }] };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "home":
        {
          content = "<div id='home-wrapper' style='padding: 20px;'><div id='new-section' class='new-wrapper'><div class='section-title'> New</div><div class='category_container'><div class='doc_category_image'></div><span class='doc_category_text'> New document</span></div></div><div id='block-wrapper'><div class='section-title'> Recent</div><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-link'></span></td><td> <span style='display: block; font-size: 14px'> Ribbon.docx</span><span style='font-size: 12px'> EJ2 >> Components >> Navigations >> Ribbon >> default</span></td></tr></tbody></table></div></div></div>";
          break;
        }
      case "new":
        {
          content = "<div id='new-section' class='new-wrapper'><div class='section-title'> New</div><div class='category_container'><div class='doc_category_image'></div> <span class='doc_category_text'> New document</span></div></div>";
          break;
        }
      case "open":
        {
          content = "<div id='open-content' style='padding: 20px;'><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-open-link'></span></td><td> <span style='display: block; font-size: 14px'> Open in Desktop App</span><span style='font-size: 12px'> Use the full functionality of Ribbon</span></td></tr></tbody></table></div><div class='section-content' style='padding: 12px 0px; cursor: pointer'><table><tbody><tr><td> <span class='doc_icon e-icons e-protect-sheet'></span></td><td> <span style='display: block; font-size: 14px'> Protect Document</span><span style='font-size: 12px'>To prevent accidental changes, this document has been set to open as view-only.</span></td></tr></tbody></table></div></div>";
          break;
        }
    }
    return content;
  }
  public menuItems: BackstageItemModel[] = [

```

```

    { id: 'home', keyTip: 'H', text: 'Home', iconCss: 'e-icons e-home',
      content: this.getBackstageContent('home') },
    { id: 'new', keyTip: 'N', text: 'New', iconCss: 'e-icons e-file-new',
      content: this.getBackstageContent('new') },
    { id: 'open', keyTip: 'O', text: 'Open', iconCss: 'e-icons e-folder-
open', content: this.getBackstageContent('open') }
  ];
  public backstageSettings: BackStageMenuModel = {
    keyTip: 'F',
    visible: true,
    items: this.menuItems,
    backButton: {
      text: 'Close',
    }
  }
  ribbonCreated(): void {
    this.ribbon.ribbonKeyTipModule.showKeyTips('F');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon #ribbon id="ribbon" (created)="ribbonCreated()"
[enableKeyTips]="true" [backStageMenu]="backstageSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>

```

```

    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

STYLES.CSS

```

@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';
/* Component custom styles */
.e-ribbon-backstage-content{
  width: 550px;
  height: 350px;
}
.section-title {
  font-size: 22px;
}
.new-docs {
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
}
.category_container {
  width: 150px;
  padding: 15px;
  text-align: center;
  cursor: pointer;
}
.doc_category_image {
  width: 80px;
  height: 100px;
  background-color: #fff;
  border: 1px solid rgb(125, 124, 124);
  text-align: center;
  overflow: hidden;
  margin: 0px auto 10px;
}
.doc_category_text {
  font-size: 16px;
}
.section-content {
  padding: 12px 0px;
  cursor: pointer;
}
.doc_icon {
  font-size: 16px;
  padding: 0px 10px;
}
.category_container:hover, .section-content:hover {

```



```

background-color: #dfdfdf;
border-radius: 5px;
transition: all 0.3s;
}

```

Ribbon layout switcher keytip

You can add keytip to the layout switcher by using the [layoutSwitcherKeyTip](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonFileMenuService, RibbonKeyTipService } from '@syncfusion/ej2-angular-ribbon'
import { Component, ViewChild } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel, RibbonSplitButtonSettingsModel, RibbonComponent } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonFileMenuService, RibbonKeyTipService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;

  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste", items: [{ text: 'Keep Source Format' }, { text: 'Merge format' }, { text: 'Keep text only' }] };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public fileMenuSettings: FileMenuSettingsModel = {
    visible: true,
    menuItems: [
      { text: "New", iconCss: "e-icons e-file-new", id: "new" },
      { text: "Open", iconCss: "e-icons e-folder-open", id: "open" },
      { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
      { text: "Save as", iconCss: "e-icons e-save", id: "save" }
    ]
  }
  ribbonCreated(): void {
    this.ribbon.ribbonKeyTipModule.showKeyTips();
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon #ribbon id="ribbon" (created)="ribbonCreated()"
[enableKeyTips]="true" layoutSwitcherKeyTip="LS"
[fileMenu]="fileMenuSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
                </e-ribbon-item>
              </e-ribbon-items>
            </e-ribbon-collection>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                </e-ribbon-item>
                <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
                </e-ribbon-item>
              </e-ribbon-items>
            </e-ribbon-collection>
          </e-ribbon-collections>
        </e-ribbon-group>
      </e-ribbon-groups>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

STYLES.CSS

```

@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';

```

Ribbon launcher icon keytip

You can add keytip to the launcher icon by using the [launcherIconKeyTip](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { RibbonModule, RibbonFileMenuService, RibbonKeyTipService } from
 '@syncfusion/ej2-angular-ribbon'
import { Component, ViewChild } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel,
 RibbonSplitButtonSettingsModel, RibbonComponent } from '@syncfusion/ej2-
 angular-ribbon';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonFileMenuService, RibbonKeyTipService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('ribbon') ribbon!: RibbonComponent;
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons
e-paste", content: "Paste", items: [{ text: 'Keep Source Format' }, { text:
'Merge format' }, { text: 'Keep text only' }] };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public fileMenuSettings: FileMenuSettingsModel = {
    visible: true,
    menuItems: [
      { text: "New", iconCss: "e-icons e-file-new", id: "new" },
      { text: "Open", iconCss: "e-icons e-folder-open", id: "open" },
      { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
      { text: "Save as", iconCss: "e-icons e-save", id: "save" }
    ]
  }
  ribbonCreated(): void {
    this.ribbon.ribbonKeyTipModule.showKeyTips('H');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon #ribbon id="ribbon" (created)="ribbonCreated()"
[enableKeyTips]="true" [fileMenu]="fileMenuSettings">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home" keyTip="H">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" [showLauncherIcon]="true"
launcherIconKeyTip="L">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>

```

```

        <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
        </e-ribbon-item>
    </e-ribbon-items>
</e-ribbon-collection>
<e-ribbon-collection>
    <e-ribbon-items>
        <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
        </e-ribbon-item>
        <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
        </e-ribbon-item>
    </e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

STYLES.CSS

```

@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';
@import 'node_modules/@syncfusion/ej2-lists/styles/material.css';
@import 'node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import 'node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import 'node_modules/@syncfusion/ej2-ribbon/styles/material.css';
@import 'node_modules/@syncfusion/ej2-angular-base/styles/material.css';

```

Methods

Show keytips

You can use the [showKeyTips](#) method to shown the keytips dynamically.

In order to show specific keytips, pass the key string as an argument in the `showKeyTips('H')` method.

Hide keytips

You can use the [hideKeyTips](#) method in Ribbon to remove the keytips dynamically. This will remove all the visible keytips.

Guidelines for adding keytips

Before adding keytips to the ribbon items consider the following:

- Avoid using the same keytip setting on multiple items.

For example: When you add the keytip text **H** or **HF** for the same items, it activates the first item occurrence of **H**, while any subsequent instances of **H** or **HF** are ignored.

- Do not use the same first letter for the single and double keytip items.

For example: When accessing keytip text **F**, **FP** and **FPF** added for the different ribbon items and pressing **F** key, only the **F** key tip associated item will be activated while the **FP**, **FPF** configured ribbon items will be ignored.

Gallery Items in Angular Ribbon component

The Ribbon supports Gallery view which allows users to perform specific actions by displaying a collection of related items, including icons, content, or images. You can render the gallery Ribbon items by using the `<e-ribbon-item>` tag directive, by specifying the `type` property to `Gallery` and customize it by using the `RibbonGallerySettingsModel`, which provides options such as `groups`, `itemCount`, `popupHeight`, `popupWidth` and more.

Groups

You can render the groups inside the gallery items by using `groups` property and customize the groups using `RibbonGalleryGroupModel`, which provides options such as `items`, `cssClass`, `header` and more.

Adding items

You can render the gallery items by using `items` property and customize using the `RibbonGalleryItemModel`, which provides options such as `content`, `iconCss`, `disabled` and more.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      items: [
        {

```

```

        content: 'Normal'
      }, {
        content: 'No Spacing'
      }, {
        content: 'Heading 1'
      }, {
        content: 'Heading 2'
      }
    ]
  }
}
];
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
      <e-ribbon-group header="Gallery" >
        <e-ribbon-collections>
          <e-ribbon-collection>
            <e-ribbon-items>

```

```

                                <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
                                </e-ribbon-item>
                                </e-ribbon-items>
                                </e-ribbon-collection>
                                </e-ribbon-collections>
                                </e-ribbon-group>
                                </e-ribbon-groups>
                                </e-ribbon-tab>
                                </e-ribbon-tabs>
</ejs-ribbon>

```

Adding content

You can use the [content](#) property to define the text content for the gallery item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel,
RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons
e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
}

```

```

    ]
  }
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>

```



```
</e-ribbon-tabs>
</ejs-ribbon>
```

Adding icons

You can use the [iconCss](#) property to define the icons for the gallery item.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel,
RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons
e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Transitions',
      items: [
        {
          content: 'None',
          iconCss: 'e-icons e-rectangle'
        }, {
          content: 'Fade',
          iconCss: 'e-icons e-send-backward'
        }, {
          content: 'Reveal',
          iconCss: 'e-icons e-bring-forward'
        }, {
          content: 'Zoom',
          iconCss: 'e-icons e-zoom-to-fit'
        }
      ]
    }
  ]
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Adding html attributes

You can use the [htmlAttributes](#) property to add HTML attributes to the Ribbon gallery item.

The following sample showcases how to add title attribute to the gallery item using `htmlAttributes` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal',
          htmlAttributes: { title: "Normal" }
        },
        {
          content: 'No Spacing',
          htmlAttributes: { title: "No Spacing" }
        },
        {
          content: 'Heading 1',
          htmlAttributes: { title: "Heading 1" }
        },
        {
          content: 'Heading 2',
          htmlAttributes: { title: "Heading 2" }
        }
      ]
    }
  ]
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

[Css class](#)

You can use the [cssClass](#) property to customize the gallery item.

The following sample showcases how to customize the appearance of each gallery item using the [cssClass](#) property .

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Good, Bad and Neutral',
      items: [{
        content: 'Normal',
        cssClass: 'normal'
      }, {
        content: 'Bad',
        cssClass: 'bad'
      }, {
        content: 'Good',
        cssClass: 'good'
      }, {
        content: 'Neutral',
        cssClass: 'neutral'
      }
    ]
  }]
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

APP.COMPONENT.CSS

```
.e-ribbon-gallery-item {
    margin: 5px;
}
.e-ribbon-gallery-item.normal{
    background: #f0f0f0;
    color: #333;
}
.e-ribbon-gallery-item.bad {
    background: #ffb6b6;
    color: #800000;
}
.e-ribbon-gallery-item.good {
    background: #c7ebc9;
    color: #004d00;
}
.e-ribbon-gallery-item.neutral {
    background: #eedd9d;
    color: #6c5429;
}
```

Disabled

You can use the [disabled](#) property to disable the Ribbon gallery item. It prevents the user interaction when set to `true`. By default, the value is `false`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
    imports: [ RibbonModule],
    providers: [ RibbonGalleryService ],
    standalone: true,
    selector: 'app-root',
    templateUrl: './app.component.html'
})
export class AppComponent {
    public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
    public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
    public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
    public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
    public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
    public gallerySettings: RibbonGallerySettingsModel = {
        groups: [{
            header: 'Styles',
            items: [
```

```

        {
            content: 'Normal',
            disabled: true
        }, {
            content: 'No Spacing'
        }, {
            content: 'Heading 1'
        }, {
            content: 'Heading 2'
        }
    ]
}]]
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>

```



```

        <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
                </e-ribbon-item>
            </e-ribbon-items>
        </e-ribbon-collection>
    </e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Custom header

You can use the [header](#) property to set header for the group items in the Ribbon gallery popup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel,
RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons
e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
}

```

```

    }
  ]
}]]
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>

```

```

    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

Setting item width

You can use the [itemWidth](#) property to specify the width of gallery items.

Setting item height

You can use the [itemHeight](#) property to set the height of the gallery items. If the [itemHeight](#) of the gallery item is smaller the remaining gallery items are aligned based on the [itemCount](#) specified.

The provided example demonstrates how to customize gallery items using the [itemWidth](#) and [itemHeight](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel,
RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons
e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    popupWidth: '350',
    groups: [{
      itemWidth: '100',
      itemHeight: '30',
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {

```

```

        content: 'Heading 2'
      }, {
        content: 'Title'
      }, {
        content: 'Subtitle'
      }
    ]
  }
}
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
      <e-ribbon-group header="Gallery" >
        <e-ribbon-collections>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-tab>
  </e-ribbon-tabs>
</ejs-ribbon>

```

```

        </e-ribbon-items>
    </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Group styling

You can use the [cssClass](#) property to customize the appearance of gallery groups.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      cssClass: "custom-group",
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
}

```

```

    }}
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```
</ejs-ribbon>
```

APP.COMPONENT.CSS

```
.custom-group {
    font-style: italic;
}
```

Setting item count

You can customize the number of items to be displayed in Ribbon gallery by using the [itemCount](#) property. By default the `itemCount` will be 3.

The following example showcases the utilization of the `itemCount` property, displaying a ribbon gallery with 4 items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel,
RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule ],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text:
"Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons
e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    itemCount: 4,
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {

```

```

        content: 'Heading 2'
    }
  ]
}]]
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
              </e-ribbon-item>
              <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
      <e-ribbon-group header="Gallery" >
        <e-ribbon-collections>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-tabs>
</ejs-ribbon>

```



```

        </e-ribbon-groups>
    </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>

```

Setting selected item

You can use the [selectedItemIndex](#) property to define the currently selected item in the Ribbon gallery items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    selectedItemIndex: 1,
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }]
  };
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>
```

Setting popup height

You can specify the height of the gallery popup by using the [popupHeight](#) property.

Setting popup width

you can specify the width of the gallery popup by using the [popupWidth](#) property.

The example demonstrates the customization of popup with `popupHeight` and `popupWidth` properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule, RibbonGalleryService } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, RibbonSplitButtonSettingsModel, RibbonButtonSettingsModel } from "@syncfusion/ej2-angular-ribbon";
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  providers: [ RibbonGalleryService ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge Format" }, { text: "Keep Text Only" }];
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteOptions, content: 'Paste' };
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content: "Format Painter" };
  public gallerySettings: RibbonGallerySettingsModel = {
    popupHeight: '180',
    popupWidth: '350',
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }, {
          content: 'Title'
        }, {
          content: 'Subtitle'
        }
      ]
    }
  ]
}
```

```

    }}
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard" groupIconCss="e-icons e-
paste">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
    <e-ribbon-group header="Gallery" >
      <e-ribbon-collections>
        <e-ribbon-collection>
          <e-ribbon-items>
            <e-ribbon-item type="Gallery"
[gallerySettings]="gallerySettings">
          </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```
</ejs-ribbon>
```

To know more about the built-in Ribbon items, please refer to the [Ribbon Items](#) section.

Help Pane

The help pane is dedicated area where the users can define help contents like controlling document permissions, sharing features, and more which appears on the right side of the Ribbon. You can use the [helpPaneTemplate](#) property to set the help pane contents.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from '@angular/core';
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
  content: "Cut" }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.CSS

```
.action_btn {
  margin: 0px 3px;
  border: none;
  color: #ffffff;
  background-color: #0d6efd;
}

#undo, #redo{
  padding: 0px 3px ;
}
```

APP.COMPONENT.HTML

```
<ejs-ribbon id="ribbon" [helpPaneTemplate]="undoredo">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
```

```

        <e-ribbon-group>
            <e-ribbon-collections>
                <e-ribbon-collection>
                    <e-ribbon-items>
                        <e-ribbon-item type="Button"
[buttonSettings]="cutButton">
                            </e-ribbon-item>
                        </e-ribbon-items>
                    </e-ribbon-collection>
                </e-ribbon-collections>
            </e-ribbon-group>
        </e-ribbon-groups>
    </e-ribbon-tab>
</e-ribbon-tabs>
<ng-template #undoredo>
    <div class="helpPaneTemplate">
        <button class="action_btn"><label><span id="undo" class="e-icons
e-undo"></span> Undo </label></button>
        <button class="action_btn"><label><span id="redo" class="e-icons
e-redo"></span> Redo </label></button>
    </div>
</ng-template>
</ejs-ribbon>

```

Tooltip

The Ribbon component supports tooltip to show additional information in the Ribbon items. The tooltip appears when the user hovers over a Ribbon item.

Adding title

You can use the [title](#) property to set the tooltip title for each Ribbon item.

```
`javascript
```

```
import { Component } from "@angular/core";
```

```
import { RibbonTooltipSettingsModel, RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
ItemModel } from '@syncfusion/ej2-angular-ribbon';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: `<!-- To Render Ribbon. -->
```

```
  <ejs-ribbon id="ribbon">
```

```
    <e-ribbon-tabs>
```

```
      <e-ribbon-tab header="Home">
```

```
        <e-ribbon-groups>
```

```
          <e-ribbon-group header="Clipboard">
```

```
            <e-ribbon-collections>
```

```
              <e-ribbon-collection>
```

```
                <e-ribbon-items>
```

```

<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteButton"
[ribbonTooltipSettings]="pasteOptions">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton" [ribbonTooltipSettings]="cutOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="copyButton" [ribbonTooltipSettings]="copyOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="formatButton"
[ribbonTooltipSettings]="formatOptions">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content:
"Format Painter" };
public pasteSettings: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge format" }, { text:
"Keep text only" }];
public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items:
this.pasteSettings, content: 'Paste' }
public cutOptions: RibbonTooltipSettingsModel = { title: "Cut" };
public copyOptions: RibbonTooltipSettingsModel = { title: "Copy" };
public formatOptions: RibbonTooltipSettingsModel = { title: "Format Painter" };

```

```
public pasteOptions: RibbonTooltipSettingsModel = { title: "Paste" };
}
`
```

Adding content

You can use the [content](#) property to set the tooltip content for each Ribbon item.

```
`javascript
import { Component } from "@angular/core";

import { RibbonTooltipSettingsModel, RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
ItemModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: 'app-root',
  templateUrl: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteButton"
[ribbonTooltipSettings]="pasteOptions">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton" [ribbonTooltipSettings]="cutOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="copyButton" [ribbonTooltipSettings]="copyOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="formatButton"
[ribbonTooltipSettings]="formatOptions">
</e-ribbon-item>
```



```

</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content:
    "Format Painter" };
  public pasteSettings: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge format" }, { text:
    "Keep text only" }];
  public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items:
    this.pasteSettings, content: 'Paste' }
  public cutOptions: RibbonTooltipSettingsModel = { title: "Cut", content: "Places the selected text or
    object on the clipboard so that you can paste it somewhere else." };
  public copyOptions: RibbonTooltipSettingsModel = { title: "Copy", content: "Copies the chosen text or
    object to the clipboard so that you can reuse it elsewhere." };
  public formatOptions: RibbonTooltipSettingsModel = { title: "Format Painter", content: "Copies the
    formatting style of a selected text or object and applies it to other content within the document." };
  public pasteOptions: RibbonTooltipSettingsModel = { title: "Paste", content: "Insert the clipboard
    content where the cursor is currently placed." };
}
`

```

Adding icon

You can use the [iconCss](#) property to specify the icons to be displayed in the tooltip.

```

`javascript
import { Component } from "@angular/core";

import { RibbonTooltipSettingsModel, RibbonButtonSettingsModel, RibbonSplitButtonSettingsModel,
  ItemModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: 'app-root',

```

```

templateUrl: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteButton"
[ribbonTooltipSettings]="pasteOptions">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton" [ribbonTooltipSettings]="cutOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="copyButton" [ribbonTooltipSettings]="copyOptions">
</e-ribbon-item>
<e-ribbon-item type="Button" [buttonSettings]="formatButton"
[ribbonTooltipSettings]="formatOptions">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };

```

```

public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };

public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-format-painter", content:
"Format Painter" };

public pasteSettings: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge format" }, { text:
"Keep text only" }];

public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items:
this.pasteSettings, content: 'Paste' }

public cutOptions: RibbonTooltipSettingsModel = { title: "Cut", content: "Places the selected text or
object on the clipboard so that you can paste it somewhere else.", iconCss: "e-icons e-cut" };

public copyOptions: RibbonTooltipSettingsModel = { title: "Copy", content: "Copies the chosen text or
object to the clipboard so that you can reuse it elsewhere.", iconCss: "e-icons e-copy" };

public formatOptions: RibbonTooltipSettingsModel = { title: "Format Painter", content: "Copies the
formatting style of a selected text or object and applies it to other content within the document.",
iconCss: "e-icons e-format-painter" };

public pasteOptions: RibbonTooltipSettingsModel = { title: "Paste", content: "Insert the clipboard
content where the cursor is currently placed.", iconCss: "e-icons e-paste" };

}
`

```

Customization

You can use the [cssClass](#) property to customize the appearance of the tooltip with your own custom styles.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RibbonModule } from '@syncfusion/ej2-angular-ribbon'
import { Component } from "@angular/core";
import { RibbonTooltipModel, RibbonButtonSettingsModel,
RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  imports: [ RibbonModule],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut",
content: "Cut" };
  public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy",
content: "Copy" };
  public formatButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-
format-painter", content: "Format Painter" };
  public pasteSettings: ItemModel[] = [{ text: "Keep Source Format" }, {
text: "Merge format" }, { text: "Keep text only" }];

```

```

    public pasteButton: RibbonSplitButtonSettingsModel = { iconCss: 'e-icons e-paste', items: this.pasteSettings, content: 'Paste' }
    public cutOptions: RibbonTooltipModel = { title: "Cut", content: "Places the selected text or object on the clipboard so that you can paste it somewhere else.", cssClass: "custom-tooltip" };
    public formatOptions: RibbonTooltipModel = { title: "Format Painter", content: "Copies the formatting style of a selected text or object and applies it to other content within the document.", cssClass: "custom-tooltip" };
    public copyOptions: RibbonTooltipModel = { title: "Copy", content: "Copies the chosen text or object to the clipboard so that you can reuse it elsewhere.", cssClass: "custom-tooltip" };
    public pasteOptions: RibbonTooltipModel = { title: "Paste", content: "Insert the clipboard content where the cursor is currently placed.", cssClass: "custom-tooltip" };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

APP.COMPONENT.HTML

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="SplitButton"
[splitButtonSettings]="pasteButton" [ribbonTooltipSettings]="pasteOptions">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="Button"
[buttonSettings]="cutButton" [ribbonTooltipSettings]="cutOptions">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="copyButton" [ribbonTooltipSettings]="copyOptions">
            </e-ribbon-item>
            <e-ribbon-item type="Button"
[buttonSettings]="formatButton" [ribbonTooltipSettings]="formatOptions">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>

```

```
</e-ribbon-tabs>
</ejs-ribbon>
```

APP.COMPONENT.CSS

```
:root {
  --borderColor: rgb(72, 72, 72);
  --black: #000000;
}
/* To customize the appearance of the tooltip */
.custom-tooltip.e-ribbon-tooltip.e-popup {
  border: 2px solid var(--borderColor);
  border-radius: 5px;
  background: var(--black);
}
/* To customize the arrow of the tooltip */
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip .e-arrow-tip-inner.e-tip-top,
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip .e-arrow-tip-inner.e-tip-bottom
{
  color: var(--black);;
}
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip-outer.e-tip-top {
  border-bottom: 8px solid var(--borderColor);
}
.custom-tooltip.e-ribbon-tooltip .e-arrow-tip-outer.e-tip-bottom {
  border-top: 8px solid var(--borderColor);;
}
/* To change the size of the tooltip title */
.custom-tooltip.e-ribbon-tooltip .e-tip-content .e-ribbon-tooltip-title {
  font-size: 14px;
}
/* To change the size of the tooltip content */
.custom-tooltip.e-ribbon-tooltip .e-tip-content .e-ribbon-text-container .e-
ribbon-tooltip-content {
  font-size: 11px;
}
```

Ribbon Resizing

The Ribbon effectively resizes the ribbon elements while being resized. It extends when the ribbon size is increased and collapses when the ribbon size is decreased. The resizing can be performed in both the classic and simplified modes. Also, we have an option to resize the ribbon elements in the custom order.

In classic mode on resizing, the items size will be changed based on the available width of the tab content from the order of Large-> Medium-> Small and viceversa.

In simplified mode on resizing, the items size will be changed based on the available width of the tab content from the order of Medium-> Small and viceversa.

Defining items allowed size

You can use the [allowedSizes](#) property to maintain a constant size for an item. If `allowedSizes` is set, it keeps the size constant even when being resized.

```
`javascript
```

```
import { Component } from "@angular/core";
```

```
import { RibbonButtonSettingsModel, RibbonItemSize, ItemModel } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: 'app-root',
  templateUrl: `<ejs-ribbon id="ribbon">
    <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
    <e-ribbon-groups>
    <e-ribbon-group>
    <e-ribbon-collections>
    <e-ribbon-collection>
    <e-ribbon-items>
    <e-ribbon-item type="SplitButton" [allowedSizes]="largeSize" [splitButtonSettings]="pasteButton">
    </e-ribbon-item>
    </e-ribbon-items>
    </e-ribbon-collection>
    <e-ribbon-collection>
    <e-ribbon-items>
    <e-ribbon-item type="Button" [allowedSizes]="mediumSize" [buttonSettings]="cutButton">
    </e-ribbon-item>
    <e-ribbon-item type="Button" [allowedSizes]="smallSize" [buttonSettings]="copyButton">
    </e-ribbon-item>
    </e-ribbon-items>
    </e-ribbon-collection>
    </e-ribbon-collections>
    </e-ribbon-group>
    </e-ribbon-groups>
    </e-ribbon-tab>
    </e-ribbon-tabs>
    </ejs-ribbon>`,
})
export class AppComponent {
  public pasteOptions: ItemModel[] = [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }];
}
```

```

public largeSize: RibbonItemSize = RibbonItemSize.Large;
public smallSize: RibbonItemSize = RibbonItemSize.Small;
public mediumSize: RibbonItemSize = RibbonItemSize.Medium;
public pasteButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-paste", content: "Paste", items:
this.pasteOptions };
public copyButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-copy", content: "Copy" };
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
}
`

```

Defining items active size

You can use the [activeSize](#) property to define the item size initially, before it is being resized. When resized the [activeSize](#) property is updated based on the ribbon's overflow state, which is determined by the [allowedSizes](#) property being configured. By default, the value is **Medium**.

Events

This section describes the ribbon events that will be triggered when appropriate actions are performed. The following events are available in the ribbon component.

tabSelected

The [tabSelected](#) event is triggered after selecting the tab item.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, TabSelectedEventArgs } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" (tabSelected)='tabSelected($event)'\>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>

```

```

</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public tabSelected(args: TabSelectedEventArgs) {
    // Your required actions here
  }
}
`

```

tabSelecting

The [tabSelecting](#) event is triggered before selecting the tab item.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, TabSelectingEventArgs } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" (tabSelecting)= 'tabSelecting($event)'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>

```



```

</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public tabSelecting(args: TabSelectingEventArgs) {
    // Your required actions here
  }
}
`

```

ribbonCollapsing

The [ribbonCollapsing](#) event is triggered before collapsing the ribbon.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, ExpandCollapseEventArgs } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" (ribbonCollapsing)= 'ribbonCollapsing($event)'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">

```

```

</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public ribbonCollapsing(args: ExpandCollapseEventArgs) {
    // Your required actions here
  }
}
`

```

ribbonExpanding

The [ribbonExpanding](#) event is triggered before expanding the ribbon.

```

`javascript
import { Component } from "@angular/core";

import { RibbonButtonSettingsModel, ExpandCollapseEventArgs } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" (ribbonExpanding)='ribbonExpanding($event)'\>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>

```

```

<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
public ribbonExpanding(args: ExpandCollapseEventArgs) {
// Your required actions here
}
}
`

```

launcherIconClick

The [launcherIconClick](#) event is triggered when the launcher icon of the group is clicked.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, LauncherClickEventArgs } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" (launcherIconClick)='launchClick($event)'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard" [showLauncherIcon]=true>
<e-ribbon-collections>
<e-ribbon-collection>

```

```

<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
public launchClick(args: LauncherClickEventArgs) {
// Your required actions here
}
}
,

```

Button item events

clicked

The [clicked](#) event is triggered when the Button is clicked.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>

```

```

<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut",
clicked: () => {
// Your required action here
} };
}
`

```

created

The [created](#) event is triggered when the Button is created.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">

```

```

<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut",
created: () => {
// Your required action here
} };
}
`

```

CheckBox item events

change

The [change](#) event is triggered when the checkbox state is changed.

```

`javascript
import { Component } from "@angular/core";
import { RibbonCheckBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>

```

```

<e-ribbon-group header="Clipboard">
  <e-ribbon-collections>
    <e-ribbon-collection>
      <e-ribbon-items>
        <e-ribbon-item type=CheckBox [checkBoxSettings]="ruler">
      </e-ribbon-item>
    </e-ribbon-items>
  </e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: false,
  change: () => {
    // Your required action here
  } };
}
`

```

created

The [created](#) event is triggered once the checkbox is created.

```

`javascript
import { Component } from "@angular/core";
import { RibbonCheckBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
  <ejs-ribbon id="ribbon">
    <e-ribbon-tabs>
      <e-ribbon-tab header="Home">

```

```

<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type=CheckBox [checkBoxSettings]="ruler">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public ruler: RibbonCheckBoxSettingsModel = { label: "Ruler", checked: false,
created: () => {
// Your required action here
} };
}
`

```

ColorPicker item events

[change](#)

The [change](#) event is triggered while changing the colors.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">

```



```

<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456', change: (args) => {
// Your required action here
}};
}
`

```

created

The [created](#) event is triggered once the ColorPicker is created.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">

```

```

<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456',
created: (args) => {
// Your required action here
}};
}
`

```

[open](#)

The [open](#) event is triggered while opening the ColorPicker popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Font">
          <e-ribbon-collection>
            <e-ribbon-items>
              <e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
            </e-ribbon-item>
          </e-ribbon-items>
        </e-ribbon-collection>
      </e-ribbon-collections>
    </e-ribbon-group>
  </e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456',
  open: (args) => {
    // Your required action here
  } };
}
`

```

select

The [select](#) event is triggered while selecting the color in picker/palette, when showButtons property is enabled.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Font">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456', select: (args) => {
      // Your required action here
    } };
  }
  ,

```

beforeClose

The [beforeClose](#) event is triggered before closing the ColorPicker popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Font">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456', beforeClose: (args) => {
      // Your required action here
    } };
  }
  ,

```

beforeOpen

The [beforeOpen](#) event is triggered before opening the ColorPicker popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Font">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456', beforeOpen: (args) => {
      // Your required action here
    } };
  }
  ,

```

beforeTileRender

The [beforeTileRender](#) event is triggered while rendering each palette tile.

```

`javascript
import { Component } from "@angular/core";
import { RibbonColorPickerSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Font">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="ColorPicker" [colorPickerSettings]="colorSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public colorSettings: RibbonColorPickerSettingsModel = { value: '#123456', beforeTileRender: (args) => {
      // Your required action here
    } };
  }
  `

```

ComboBox item events

change

The [change](#) event is triggered when an item in a popup is selected or when the model value is changed by the user.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",

```

```

template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];
public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
'150px', allowFiltering: true,
change: (args) => {
// Your required action here
} }
}
`

```

[close](#)

The [close](#) event is triggered when the popup is closed.

`javascript


```

import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];

  public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
    '150px', allowFiltering: true,

  close: (args) => {
    // Your required action here
  } };
}

```

`

[open](#)

The [open](#) event is triggered when the popup is opened.

`javascript

```
import { Component } from "@angular/core";
```

```
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
```

```
@Component({
```

```
  selector: "app-root",
```

```
  template: `<!-- To Render Ribbon. -->
```

```
  <ejs-ribbon id="ribbon">
```

```
    <e-ribbon-tabs>
```

```
      <e-ribbon-tab header="Home">
```

```
        <e-ribbon-groups>
```

```
          <e-ribbon-group header="Font">
```

```
            <e-ribbon-collections>
```

```
              <e-ribbon-collection>
```

```
                <e-ribbon-items>
```

```
                  <e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
```

```
                </e-ribbon-item>
```

```
              </e-ribbon-items>
```

```
            </e-ribbon-collection>
```

```
          </e-ribbon-collections>
```

```
        </e-ribbon-group>
```

```
      </e-ribbon-groups>
```

```
    </e-ribbon-tab>
```

```
  </e-ribbon-tabs>
```

```
</ejs-ribbon>`,
```

```
  })
```

```
export class AppComponent {
```

```
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];
```

```
  public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
    '150px', allowFiltering: true,
```

```

open: (args) => {
// Your required action here
} };
}
`

```

created

The [created](#) event is triggered when the popup is Created.

`javascript

```

import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {

```

```

public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
"Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
Roman", "Verdana", "Windings"];

public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
'150px', allowFiltering: true,
created: (args) => {
// Your required action here
} };
```

filtering

The [filtering](#) event triggers on typing a character in the combobox.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
```

```

</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];
  public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
    '150px', allowFiltering: true,
  filtering: (args) => {
    // Your required action here
  } };
}
`

```

select

The [select](#) event is triggered when an item in the popup is selected.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
  <ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
  <e-ribbon-tab header="Home">
  <e-ribbon-groups>
  <e-ribbon-group header="Font">
  <e-ribbon-collections>
  <e-ribbon-collection>
  <e-ribbon-items>
  <e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
  </e-ribbon-item>
  </e-ribbon-items>
  </e-ribbon-collection>

```

```

</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];

  public fontStyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
    '150px', allowFiltering: true,
  select: (args) => {
    // Your required action here
  } };
}
`

```

beforeOpen

The [beforeOpen](#) event triggers before opening the popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonComboBoxSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Font">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>

```

```

<e-ribbon-item type="ComboBox" [comboBoxSettings]="fontstyleSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public fontStyle: string[] = ["Algerian", "Arial", "Calibri", "Cambria", "Cambria Math", "Courier New",
    "Candara", "Georgia", "Impact", "Segoe Print", "Segoe Script", "Segoe UI", "Symbol", "Times New
    Roman", "Verdana", "Windings"];
  public fontstyleSettings: RibbonComboBoxSettingsModel = { dataSource: this.fontStyle, index: 3, width:
    '150px', allowFiltering: true,
  beforeOpen: (args) => {
    // Your required action here
  } };
}
`

```

DropDown item events

beforeClose

The [beforeClose](#) event is triggered before closing the DropDownButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
  <ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
  <e-ribbon-tab header="Home">

```

```

<e-ribbon-groups>
  <e-ribbon-group header="Tables">
    <e-ribbon-collections>
      <e-ribbon-collection>
        <e-ribbon-items>
          <e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
        </e-ribbon-item>
        </e-ribbon-items>
      </e-ribbon-collection>
    </e-ribbon-collections>
  </e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, beforeClose: (args) => {
      // Your required action here
    } };
  }
`

```

beforeOpen

The [beforeOpen](#) event is triggered before opening the DropDownButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```



```

<ejs-ribbon id="ribbon">
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Tables">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
  })
  export class AppComponent {
    public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
    public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, beforeOpen: (args) => {
      // Your required action here
    } };
  }
  ,

```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each popup item of the DropDownButton.

```

`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';

```

```

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
    <ejs-ribbon id="ribbon">
      <e-ribbon-tabs>
        <e-ribbon-tab header="Home">
          <e-ribbon-groups>
            <e-ribbon-group header="Tables">
              <e-ribbon-collections>
                <e-ribbon-collection>
                  <e-ribbon-items>
                    <e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
                  </e-ribbon-item>
                </e-ribbon-items>
              </e-ribbon-collection>
            </e-ribbon-collections>
          </e-ribbon-group>
        </e-ribbon-groups>
      </e-ribbon-tab>
    </e-ribbon-tabs>
  </ejs-ribbon>`,
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, beforeItemRender: (args) => {
    // Your required action here
  } };
}
`

```

[open](#)

The [open](#) event is triggered while opening the DropDownButton popup.

`javascript

```

import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Tables">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, open: (args) => {
    // Your required action here
  } };
}
`

```

close

The [close](#) event is triggered while closing the DropDownButton popup.

```
`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Tables">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table",
  items: this.tableOptions, close: (args) => {
    // Your required action here
  }
}
```

```

});
}
,

```

created

The [created](#) event is triggered when the DropDown is created.

```

`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Tables">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];

```

```

public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table",
items: this.tableOptions, close: (args) => {
// Your required action here
} };
}
`

```

[select](#)

The [select](#) event is triggered while selecting an action item in the DropDownButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonDropDownSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { ItemModel } from '@syncfusion/ej2-angular-splitbuttons';
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Tables">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="DropDown" [dropDownSettings]="tableSettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})

```

```
export class AppComponent {
  public tableOptions: ItemModel[] = [{ text: "Insert Table" }, { text: "This device" }, { text: "Convert Table" }, { text: "Excel SpreadSheet" }];
  public tableSettings: RibbonDropDownSettingsModel = { iconCss: "e-icons e-table", content: "Table", items: this.tableOptions, select: (args) => {
    // Your required action here
  } };
}
```

SplitButton item events

beforeClose

The [beforeClose](#) event is triggered before closing the SplitButton popup.

`javascript

```
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
```

```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  beforeClose: () => {
    // Your required action here
  } };
}
`

```

beforeOpen

The [beforeOpen](#) event is triggered before opening the SplitButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```



```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  beforeOpen: () => {
    // Your required action here
  } };
}
`

```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each popup item of SplitButton.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep
Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  beforeItemRender: () => {
    // Your required action here
  } };
}
`

```

[open](#)

The [open](#) event is triggered while opening the SplitButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep
Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  open: () => {
    // Your required action here
  } };
}
`

```

close

The [close](#) event is triggered while opening the SplitButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep
Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  close: () => {
    // Your required action here
  } };
}
`

```

created

The [created](#) event is triggered when the SplitButton is created.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>`,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  created: () => {
    // Your required action here
  } };
}
`

```

select

The [select](#) event is triggered while selecting an action item in the SplitButton popup.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>`,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  select: (args) => {
    // Your required action here
  } };
}
`

```

[click](#)

The [click](#) event is triggered while clicking the primary button in the SplitButton.

```

`javascript
import { Component } from "@angular/core";
import { RibbonSplitButtonSettingsModel } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="SplitButton" [splitButtonSettings]="pasteSettings"></e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>

```

```

</ejs-ribbon>,
})
export class AppComponent {
  public pasteSettings: RibbonSplitButtonSettingsModel = { iconCss: "e-icons e-paste", items: [{ text: "Keep Source Format" }, { text: "Merge format" }, { text: "Keep text only" }], content: "Paste",
  click: (args) => {
    // Your required action here
  } };
}
`

```

GroupBox item events

beforeClick

The [beforeClick](#) event is triggered before selecting a button from the groupbutton items.

```

`javascript
import { Component } from "@angular/core";

import { RibbonItemSize, RibbonGroupBoxSettingsModel, RibbonGroupBoxSelection,
BeforeClickGroupBoxEventArgs } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  templateUrl: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Paragraph" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="GroupBox" [allowedSizes]="smallSize" [groupBoxSettings]="events">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>

```

```

</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public events: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Multiple,
    items: [
      {iconCss: 'e-icons e-bold', content: 'Bold',
        beforeClick: (args: BeforeClickGroupButtonEventArgs) => {
          // Your required action here
        }},
      {iconCss: 'e-icons e-italic', content: 'Italic',
        beforeClick: (args: BeforeClickGroupButtonEventArgs) => {
          // Your required action here
        }, selected: true},
      {iconCss: 'e-icons e-underline', content: 'Underline',
        beforeClick: (args: BeforeClickGroupButtonEventArgs) => {
          // Your required action here
        }},
      {iconCss: 'e-icons e-strikethrough', content: 'Strikethrough',
        beforeClick: (args: BeforeClickGroupButtonEventArgs) => {
          // Your required action here
        }}
    ]
  }
  public smallSize: RibbonItemSize = RibbonItemSize.Small;
}
`

```

click

The [click](#) event is triggered when selecting a button from the groupbutton items.

`javascript


```

import { Component } from "@angular/core";
import { RibbonItemSize, RibbonGroupButtonSettingsModel, RibbonGroupButtonSelection,
ClickGroupButtonEventArgs } from '@syncfusion/ej2-angular-ribbon';
@Component({
  selector: "app-root",
  templateUrl: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Paragraph" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="GroupButton" [allowedSizes]="smallSize" [groupButtonSettings]="events">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public events: RibbonGroupButtonSettingsModel = {
    selection: RibbonGroupButtonSelection.Single,
    items: [
      {iconCss: 'e-icons e-align-left', content: 'Align Left',
        click:(args: ClickGroupButtonEventArgs) => {
          // Your required action here
        }},
    ]
  }
}

```

```

{iconCss: 'e-icons e-align-center', content: 'Align Center',
click:(args: ClickGroupButtonEventArgs) => {
// Your required action here
}, selected: true},
{iconCss: 'e-icons e-align-right', content: 'Align Right',
click:(args: ClickGroupButtonEventArgs) => {
// Your required action here
}},
{iconCss: 'e-icons e-justify', content: 'Justify',
click:(args: ClickGroupButtonEventArgs) => {
// Your required action here
}}
]
}

public smallSize: RibbonItemSize = RibbonItemSize.Small;
}
`

```

FileMenu events

beforeClose

The [beforeClose](#) event is triggered before closing the fileMenu popup.

`javascript

```

import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>

```

```

<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
{ text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
{ text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
{ text: "Save as", iconCss: "e-icons e-save", id: "save" }]
public fileSettings: FileMenuSettingsModel = {
menuItems: this.fileOptions,
visible: true,
beforeClose: () => {
// Your required action here
}
};
}
`

```

beforeOpen

The [beforeOpen](#) event is triggered before Opening the fileMenu popup.

`javascript

```

import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";

```

```

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button" [buttonSettings]="cutButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" } ]
  public fileSettings: FileMenuSettingsModel = {
    menuItems: this.fileOptions,
    visible: true,
    beforeOpen: () => {
      // Your required action here
    }
  }
}

```

```

}
};
}
`

```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each ribbon fileMenu item.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {

```

```

public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
{ text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
{ text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
{ text: "Save as", iconCss: "e-icons e-save", id: "save" }]
public fileSettings: FileMenuSettingsModel = {
  menuItems: this.fileOptions,
  visible: true,
  beforeItemRender: () => {
    // Your required action here
  }
};

```

[open](#)

The [open](#) event is triggered when the fileMenu popup is opened.

`javascript

```

import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>

```

```

</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
  { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
  { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  public fileSettings: FileMenuSettingsModel = {
    menuItems: this.fileOptions,
    visible: true,
    open: () => {
      // Your required action here
    }
  };
}
`

```

close

The [close](#) event is triggered when the fileMenu popup is closed.

```

`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->

```

```

<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
  <e-ribbon-tabs>
    <e-ribbon-tab header="Home">
      <e-ribbon-groups>
        <e-ribbon-group header="Clipboard">
          <e-ribbon-collections>
            <e-ribbon-collection>
              <e-ribbon-items>
                <e-ribbon-item type="Button" [buttonSettings]="cutButton">
              </e-ribbon-item>
            </e-ribbon-items>
          </e-ribbon-collection>
        </e-ribbon-collections>
      </e-ribbon-group>
    </e-ribbon-groups>
  </e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public fileOptions: MenuitemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
    { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
    { text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
    { text: "Save as", iconCss: "e-icons e-save", id: "save" }]
  public fileSettings: FileMenuSettingsModel = {
    menuItems: this.fileOptions,
    visible: true,
    close: () => {
      // Your required action here
    }
  };
}

```


select

The [select](#) event is triggered while selecting an item in the ribbon fileMenu.

```
`javascript
import { Component } from "@angular/core";
import { RibbonButtonSettingsModel, FileMenuSettingsModel } from '@syncfusion/ej2-angular-ribbon';
import { MenuItemModel } from "@syncfusion/ej2-navigations";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [fileMenu]='fileSettings'>
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public fileOptions: MenuItemModel[] = [{ text: "New", iconCss: "e-icons e-file-new", id: "new" },
  { text: "Open", iconCss: "e-icons e-folder-open", id: "Open" },
```

```

{ text: "Rename", iconCss: "e-icons e-rename", id: "rename" },
{ text: "Save as", iconCss: "e-icons e-save", id: "save" }]
public fileSettings: FileMenuSettingsModel = {
  menuItems: this.fileOptions,
  visible: true,
  select: () => {
    // Your required action here
  }
};
}
,

```

Backstage Menu events

[backStageItemClick](#)

The [backStageItemClick](#) event is triggered when backstage item is selected.

`javascript

```

import { Component } from "@angular/core";

import { RibbonButtonSettingsModel, BackStageMenuModel, BackstageItemModel,
  BackstageItemClickArgs } from '@syncfusion/ej2-angular-ribbon';

@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon" [backStageMenu]="backstageSettings">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Clipboard">
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Button" [buttonSettings]="cutButton">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>

```

```

</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public cutButton: RibbonButtonSettingsModel = { iconCss: "e-icons e-cut", content: "Cut" };
  public backstageSettings: BackStageMenuModel = {
    text: 'File',
    visible: true,
    items: this.menuItems,
    backButton: {
      text: 'Close',
    }
  }
  public getBackstageContent(item: string): string {
    var content = "";
    switch (item) {
      case "new":
        {
          content = "<div id='new-section' class='new-wrapper'><div class='section-title'> New </div><div class='categorycontainer'><div class='doccategoryimage'></div> <span class='doccategory_text'> New document </span></div></div>";
        }
      }
    return content;
  }
  public menuItems: BackstageItemModel[] = [{
    id: 'new',
    text: 'New',
    iconCss: 'e-icons e-file-new',
    content: this.getBackstageContent('new'),
    backstageItemClick: (args: BackstageItemClickArgs) => {

```

```
// Your required action here
}
});
}
`
```

Gallery events

popupOpen

The [popupOpen](#) event is triggered when the gallery popup opens.

```
`javascript
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, GalleryPopupEventArgs } from "@syncfusion/ej2-angular-ribbon";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Gallery" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
```

```

public gallerySettings: RibbonGallerySettingsModel = {
  groups: [{
    header: 'Styles',
    items: [
      {
        content: 'Normal'
      }, {
        content: 'No Spacing'
      }, {
        content: 'Heading 1'
      }, {
        content: 'Heading 2'
      }
    ]
  }],
  popupOpen: (args: GalleryPopupEventArgs) => {
    // Your required action here
  }
};

```

popupClose

The [popupClose](#) event is triggered when the gallery popup closes.

`javascript

```

import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, GalleryPopupEventArgs } from "@syncfusion/ej2-angular-ribbon";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>

```

```

<e-ribbon-group header="Gallery" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public gallerySettings: RibbonGallerySettingsModel = {
groups: [{
header: 'Styles',
items: [
{
content: 'Normal'
}, {
content: 'No Spacing'
}, {
content: 'Heading 1'
}, {
content: 'Heading 2'
}
]
}],
popupClose: (args: GalleryPopupEventArgs) => {
// Your required action here

```

```

}
};
}
`

```

itemHover

The [itemHover](#) event is triggered when hover over the gallery item.

```

`javascript
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, GalleryHoverEventArgs } from "@syncfusion/ej2-angular-ribbon";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Gallery" >
<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public gallerySettings: RibbonGallerySettingsModel = {

```

```

groups: [{
  header: 'Styles',
  items: [
    {
      content: 'Normal'
    }, {
      content: 'No Spacing'
    }, {
      content: 'Heading 1'
    }, {
      content: 'Heading 2'
    }
  ]
}],
itemHover: (args: GalleryHoverEventArgs) => {
  // Your required action here
}
};
}
,

```

beforeItemRender

The [beforeItemRender](#) event is triggered while rendering each gallery item.

```

`javascript
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, GalleryItemEventArgs } from "@syncfusion/ej2-angular-ribbon";
@Component({
  selector: "app-root",
  template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Gallery" >

```



```

<e-ribbon-collections>
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
public gallerySettings: RibbonGallerySettingsModel = {
groups: [{
header: 'Styles',
items: [
{
content: 'Normal'
}, {
content: 'No Spacing'
}, {
content: 'Heading 1'
}, {
content: 'Heading 2'
}
]
}],
beforeItemRender: (args: GalleryItemEventArgs) => {
// Your required action here
}
}

```

```
};
}
,
```

beforeSelect

The [beforeSelect](#) event is triggered before selecting an item in the Ribbon gallery.

```
`javascript
```

```
import { Component } from "@angular/core";
```

```
import {RibbonGallerySettingsModel, GalleryBeforeSelectEventArgs } from "@syncfusion/ej2-angular-ribbon";
```

```
@Component({
```

```
  selector: "app-root",
```

```
  template: `<!-- To Render Ribbon. -->
```

```
  <ejs-ribbon id="ribbon">
```

```
    <e-ribbon-tabs>
```

```
      <e-ribbon-tab header="Home">
```

```
        <e-ribbon-groups>
```

```
          <e-ribbon-group header="Gallery" >
```

```
            <e-ribbon-collections>
```

```
              <e-ribbon-collection>
```

```
                <e-ribbon-items>
```

```
                  <e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
```

```
                </e-ribbon-item>
```

```
              </e-ribbon-items>
```

```
            </e-ribbon-collection>
```

```
          </e-ribbon-collections>
```

```
        </e-ribbon-group>
```

```
      </e-ribbon-groups>
```

```
    </e-ribbon-tab>
```

```
  </e-ribbon-tabs>
```

```
  </ejs-ribbon>`,
```

```
  })
```

```
  export class AppComponent {
```

```
    public gallerySettings: RibbonGallerySettingsModel = {
```

```
    groups: [{
```

```

header: 'Styles',
items: [
{
content: 'Normal'
}, {
content: 'No Spacing'
}, {
content: 'Heading 1'
}, {
content: 'Heading 2'
}
]
}],
beforeSelect: (args: GalleryBeforeSelectEventArgs) => {
// Your required action here
}
};
}
,

```

select

The [select](#) event is triggered while selecting an item in the Ribbon Gallery.

```

`javascript
import { Component } from "@angular/core";
import { RibbonGallerySettingsModel, GallerySelectEventArgs } from "@syncfusion/ej2-angular-ribbon";
@Component({
selector: "app-root",
template: `<!-- To Render Ribbon. -->
<ejs-ribbon id="ribbon">
<e-ribbon-tabs>
<e-ribbon-tab header="Home">
<e-ribbon-groups>
<e-ribbon-group header="Gallery" >
<e-ribbon-collections>

```

```
<e-ribbon-collection>
<e-ribbon-items>
<e-ribbon-item type="Gallery" [gallerySettings]="gallerySettings">
</e-ribbon-item>
</e-ribbon-items>
</e-ribbon-collection>
</e-ribbon-collections>
</e-ribbon-group>
</e-ribbon-groups>
</e-ribbon-tab>
</e-ribbon-tabs>
</ejs-ribbon>`,
})
export class AppComponent {
  public gallerySettings: RibbonGallerySettingsModel = {
    groups: [{
      header: 'Styles',
      items: [
        {
          content: 'Normal'
        }, {
          content: 'No Spacing'
        }, {
          content: 'Heading 1'
        }, {
          content: 'Heading 2'
        }
      ]
    }
  ]
},
select: (args: GallerySelectEventArgs) => {
  // Your required action here
}
};
```

```
}
,
```

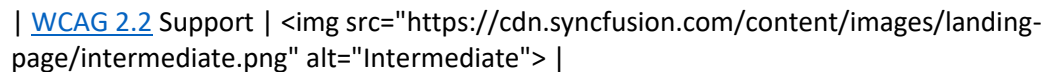
Accessibility in Angular Ribbon component

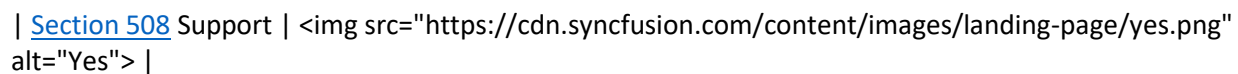
The Ribbon component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

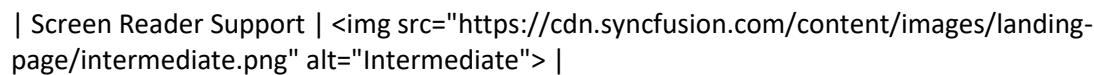
The accessibility compliance for the Ribbon component is outlined below.

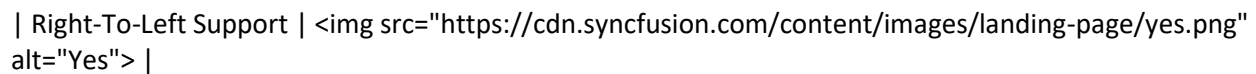
| Accessibility Criteria | Compatibility |

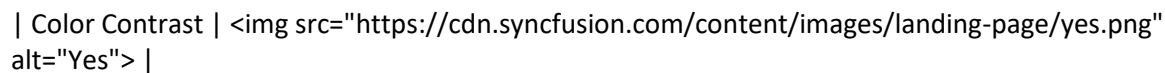
| -- | -- |

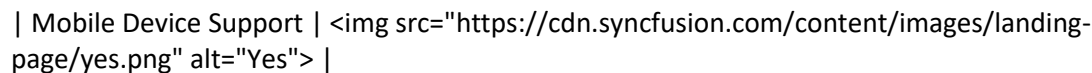
| [WCAG 2.2](#) Support |  |

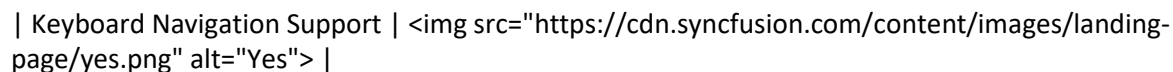
| [Section 508](#) Support |  |

| Screen Reader Support |  |

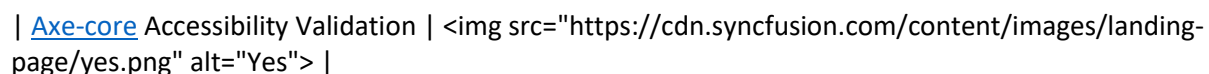
| Right-To-Left Support |  |

| 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 Ribbon component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Ribbon component:

Attributes	Purpose
---	---
<code>role=tablist</code>	Used to identify the element that serves as the container for a set of tabs.
<code>role=tab</code>	Indicates an interactive element within a tablist that, when activated, displays its associated tab panel.
<code>role=tabpanel</code>	Specifies the role for the content associated with an active tab, describing its role in presenting the active content.
<code>role=button</code>	Represents a clickable element that trigger a response when activated by the user.
<code>role=menu</code>	Represents an item that have sub menu.
<code>role=menuitem</code>	Indicates an option in a set of choices within a menu.
<code>role=combobox</code>	Identifies an element as an input that controls another element, commonly used for dropdowns.
<code>role=option</code>	Used for selectable items in a combobox.
<code>role=gridcell</code>	Specified as gridcell for the tiles in the color palette.
<code>aria-orientation</code>	Indicates the element's orientation as horizontal, vertical, or unknown/ambiguous.
<code>aria-selected</code>	Indicates the current <code>selected</code> state of various widgets.
<code>aria-labelledby</code>	Sets to the Tab content element to indicates the associated Tab header for the content.
<code>aria-controls</code>	Indicates the associated tabpanel for the header by setting the attribute on Tab items.
<code>aria-haspopup</code>	Indicates availability and type of interactive popup triggered by the element it's set on.
<code>aria-disabled</code>	Indicates that the element is perceivable but disabled, making it not editable or operable.
<code>aria-expanded</code>	Indicates whether a component is expanded or collapsed, set on the respective element.
<code>aria-label</code>	Defines a string value that labels an interactive element for accessibility.
<code>aria-checked</code>	Indicates the current <code>checked</code> state of checkboxes, radio buttons, and other widgets.

| **aria-owns** | Identifies an element or elements, establishing a relationship when DOM hierarchy can't represent it. |

| **aria-readonly** | Indicates that the element is not editable but is otherwise operable. |

| **aria-activedescendent** | Identifies the currently active element when focus is on a combobox, textbox, group, or application. |

| **aria-autocomplete** | Indicates whether inputting text could trigger display of predictions and specifies how predictions will be presented for a combobox, searchbox, or textbox. |

Keyboard interaction

The Ribbon 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 Ribbon component.

| **Press** | **To do this** |

| --- | --- |

Ribbon Tab||

| **Tab** | **To focus the ribbon tabs.** |

| **Right Arrow** | **Moves focus to the next Tab.** |

| **Left Arrow** | **Moves focus to the previous Tab.** |

| **Enter / Space** | **To select the currently focused ribbon tab.** |

Ribbon Items||

| **Tab** | **To focus the ribbon Items.** |

| **Right Arrow** | **Focuses the next item.** |

| **Left Arrow** | **Focuses the previous item.** |

| **Enter / Space** | **To select the currently focused ribbon item.** |

Ribbon Dropdown Items/ Ribbon Split button||

| **Esc** | **Closes the popup.** |

| **Enter / Space** | **Opens the popup, or activates the highlighted item and closes the popup.** |

| **Arrow Up** | **Focuses the next item.** |

| **Arrow Down** | **Focuses the previous item.** |

| **Alt + Arrow Up** | **Closes the popup.** |

| **Alt + Arrow Down** | **Opens the popup** |

Ribbon File menu||

| **Tab** | **To focus the ribbon file menu.** |

| **Esc** | **Closes the popup.** |

| **Enter** | **Opens the popup, or activates the highlighted item and closes the popup.** |

| Arrow Up | Focuses the previous action item. |

| Arrow Down | Focuses the next action item. |

| Alt + Arrow Down | Opens the popup |

Ribbon Combobox |

| 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 |

Ensuring accessibility

The Ribbon component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Ribbon component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Ribbon component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

RichTextEditor

Getting started with Angular Rich text editor component

This section explains the steps required to create a simple Angular Rich Text Editor component and configure its available functionalities.

Setup Angular Environment

Use [Angular CLI](#) to setup the Angular applications. To install Angular CLI, use the following command.

```
`javascript
```



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

```
,
```

Create an Angular Application

Start a new Angular application using the following Angular CLI command.

```
`javascript
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion Rich Text Editor package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. Get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components. They are:

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages($\geq 20.2.36$) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the following command.

Add [@syncfusion/ej2-angular-richtexteditor](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-richtexteditor --save
```

```
,
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package, use the following command.

Add [@syncfusion/ej2-angular-richtexteditor@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-richtexteditor@ngcc --save
```

```
,
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as follows.

```
`bash
```

```
@syncfusion/ej2-angular-richtexteditor:"20.2.38-ngcc"
```

```
,
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Rich Text Editor module

Import Rich Text Editor module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-richtexteditor` [src/app/app.module.ts].

```
`javascript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
// Imported Syncfusion RichTextEditorModule from Rich Text Editor package
import { RichTextEditorModule } from '@syncfusion/ej2-angular-richtexteditor';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    // Registering EJ2 Rich Text Editor Module
    RichTextEditorModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Adding CSS reference

Add Rich Text Editor component's styles as given in the following `styles.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-icons/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-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';
`
```

Adding Rich Text Editor component

Modify the template in the [src/app/app.component.ts] file to render the Rich Text Editor component. Add the Angular Rich Text Editor by using the `<ejs-richtexteditor>` selector in the `template` section of the app.component.ts file.

```
`typescript
import { Component } from '@angular/core';

import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';

@Component({
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE'>
<ng-template #valueTemplate>
<p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides
the best user experience to create and update the content.

Users can format their content using standard toolbar commands.</p>
<p><b>Key features:</b></p>
<ul><li><p>Provides <IFRAME> and <DIV> modes.</p></li>
<li><p>Capable of handling markdown editing.</p></li>
<li><p>Contains a modular library to load the necessary functionality on demand.</p></li>
<li><p>Provides a fully customizable toolbar.</p></li>
<li><p>Provides HTML view to edit the source directly for developers.</p></li>
<li><p>Supports third-party library integration.</p></li>
<li><p>Allows preview of modified content before saving it.</p></li>
<li><p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p></li>
<li><p>Contains undo/redo manager. </p></li>
<li><p>Creates bulleted and numbered lists.</p></li>
</ul>
</ng-template>
</ejs-richtexteditor>`,
```

providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]

}}

export class AppComponent {

}

,

Initialize Rich Text Editor from ``<iframe>`` element

The Rich Text Editor's content is placed in an `iframe` and isolated from the rest of the page.

Initialize the Rich Text Editor on `div` element and set the enable field `iframeSettings` property to true.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='iframeRTE' [toolbarSettings]='tools' [iframeSettings]='iframe' [height]='height'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor
        that provides the best user experience to create and update the content.
        Users can format their content using standard toolbar
        commands.</p>
        <p><b>Key features:</b></p>
        <ul><li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;
        modes</p></li>
        <li><p>Capable of handling markdown editing.</p></li>
        <li><p>Contains a modular library to load the necessary
        functionality on demand.</p></li>
        <li><p>Provides a fully customizable toolbar.</p></li>
        <li><p>Provides HTML view to edit the source directly for
        developers.</p></li>
        <li><p>Supports third-party library integration.</p></li>
        <li><p>Allows preview of modified content before saving
        it.</p></li>
        <li><p>Handles images, hyperlinks, video, hyperlinks, uploads,
        etc.</p></li>
        <li><p>Contains undo/redo manager.</p></li>
        <li><p>Creates bulleted and numbered lists.</p></li>
      </ng-template>
    </ejs-richtexteditor>`
})
```

```

        </ul>
    </ng-template>
</ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
    public tools: object = {
        items: ['Undo', 'Redo', '|',
            'Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
            'FontName', 'FontSize', 'FontColor', 'BackgroundColor', '|',
            'SubScript', 'SuperScript', '|',
            'LowerCase', 'UpperCase', '|',
            'Formats', 'Alignments', '|', 'OrderedList', 'UnorderedList',
            '|',
            'Indent', 'Outdent', '|', 'CreateLink',
            'Image', '|', 'ClearFormat', 'Print', 'SourceCode', '|',
            'FullScreen']
    };
    public iframe: object = { enable: true };
    public height: number = 500;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Module Injection

To create Rich Text Editor with additional features, inject the required modules. The following modules are used to extend Rich Text Editor's basic functionality.

- **Toolbar** - Inject this module to use Toolbar feature.
- **Link** - Inject this module to use link feature in Rich Text Editor.
- **Image** - Inject this module to use image feature in Rich Text Editor.
- **Table** - Inject this module to use table feature in Rich Text Editor.
- **Count** - Inject this module to use character count in Rich Text Editor.
- **HtmlEditor** - Inject this module to use Rich Text Editor as html editor.
- **MarkdownEditor** - Inject this module to use Rich Text Editor as markdown editor.
- **QuickToolbar** - Inject this module to use quick toolbar feature for the target element.
- **Resize** - Inject this module to use resize feature in Rich Text Editor.
- **FileManager** - Inject this module to use file browser feature in Rich Text Editor.
- **PasteCleanup** - Inject this module to use paste cleanup feature in Rich Text Editor.
- **FormatPainter** - Inject this module to use format painter feature in Rich Text Editor.
- **EmojiPicker** - Inject this module to use emoji picker feature in Rich Text Editor.

These modules should be injected into the provider section of **AppModule**.

Run the application

Use the following command to run the application in the browser.

```
`bash
```

```
ng serve --open
```

```
`
```

Output will appear as follows.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE'>
<ng-template #valueTemplate>
<p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor that provides the best user experience to create and update the content.
  Users can format their content using standard toolbar commands.</p>
<p><b>Key features:</b></p>
<ul><li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes.</p></li>
<li><p>Capable of handling markdown editing.</p></li>
<li><p>Contains a modular library to load the necessary functionality on demand.</p></li>
<li><p>Provides a fully customizable toolbar.</p></li>
<li><p>Provides HTML view to edit the source directly for developers.</p></li>
<li><p>Supports third-party library integration.</p></li>
<li><p>Allows preview of modified content before saving it.</p></li>
<li><p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p></li>
<li><p>Contains undo/redone manager. </p></li>
<li><p>Creates bulleted and numbered lists.</p></li>
</ul>
</ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Configure the Toolbar

Configure the toolbar with custom tools using items field of toolbarSettings property in your application.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='rteTool' [toolbarSettings]='tools'>
<ng-template #valueTemplate>
<p>The Rich Text Editor triggers events based on its actions. </p>
<p>The events can be used as an extension point to perform custom
operations.</p>
<ul>
<li>created - Triggers when the component is rendered.</li>
<li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
<li>focus - Triggers when RTE is focused in.</li>
<li>blur - Triggers when RTE is focused out.</li>
<li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
<li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
<li>destroyed - Triggers when the component is destroyed.</li>
</ul>
</ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public tools: object = {
    items: ['Undo', 'Redo', '|',
      'Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor', '|',
      'SubScript', 'SuperScript', '|',
      'LowerCase', 'UpperCase', '|'],
```

```

        'Formats', 'Alignments', '|', 'OrderedList', 'UnorderedList', '|',
        'Indent', 'Outdent', '|', 'CreateLink',
        'Image', '|', 'ClearFormat', 'Print', 'SourceCode', '|',
        'FullScreen']
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Insert images and links

The image module inserts an image into Rich Text Editor's content area, and the link module links external resources such as website URLs, to selected text in the Rich Text Editor's content, respectively.

The link inject module adds a link icon to the toolbar and the image inject module adds an image icon to the toolbar.

Specifies the items to be rendered in the quick toolbar based on the target element such image, link, and text element. The quick toolbar opens to customize the element by clicking the target element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, QuickToolbarService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools' [quickToolbarSettings]='quickTools'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor triggers events based on its actions. </p>
      <p>The events can be used as an extension point to perform custom operations.</p>
      <ul>
        <li>created - Triggers when the component is rendered.</li>
        <li>change - Triggers only when RTE is blurred and changes are done to the content.</li>
        <li>focus - Triggers when RTE is focused in.</li>
        <li>blur - Triggers when RTE is focused out.</li>
        <li>actionBegin - Triggers before command execution using toolbar items or executeCommand method.</li>

```



```

        <li>actionComplete - Triggers after command execution using toolbar
        items or executeCommand method.</li>
        <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
    public tools: object = {
        items: ['Undo', 'Redo', '|',
            'Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
            'FontName', 'FontSize', 'FontColor', 'BackgroundColor', '|',
            'SubScript', 'SuperScript', '|',
            'LowerCase', 'UpperCase', '|',
            'Formats', 'Alignments', '|', 'OrderedList', 'UnorderedList',
            '|',
            'Indent', 'Outdent', '|', 'CreateLink',
            'Image', '|', 'ClearFormat', 'Print', 'SourceCode', '|',
            'FullScreen']
    };
    public quickTools: object = {
        image: [
            'Replace', 'Align', 'Caption', 'Remove', 'InsertLink', '-',
            'Display', 'AltText', 'Dimension']
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Retrieve the formatted content

To retrieve the editor contents, use the value property of Rich Text Editor.

```
`typescript
```

```
let rteValue: string = this.rteObj.value;
```

```
,
```

Or, you can use the [getHtml](#) public method to retrieve the Rich Text Editor content.

```
`typescript
```

```
let rteValue: string = this.rteObj.getHtml();
```

```
,
```

To fetch the Rich Text Editor's text content, use [getText](#) method.

```
`typescript
```

```
let rteValue: string = this.rteObj.contentModule.getText();
```

Retrieve the number of characters in the Rich Text Editor

To get the maximum number of characters in the Rich Text Editor's content, use [getCharCount](#)

```
`typescript
```

```
let rteCount: number = this.rteObj.getCharCount();
```

You can refer to our [Angular Rich Text Editor](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Rich Text Editor example](#) to know how to render and configure the rich text editor tools.

See Also

- [How to change the editor type](#)
- [How to render the iframe](#)
- [How to render the toolbar in inline mode](#)
- [How to insert Emoticons](#)
- [Blog posting using Rich Text Editor](#)
- [Reactive Form with Rich Text Editor](#)

Editor mode in Angular Rich text editor component

The Rich Text Editor component used to create, edit and return the content in valid HTML markup or markdown (MD) of the content. It supports following two editing formation.

- HTML Editor
- Markdown Editor

HTML Editor

Rich Text Editor is a WYSIWYG editing control for formatting the word content as HTML.

The HTML editing mode is the default mode of Rich Text Editor. Which is used for format the content through the available toolbar items and returns the valid HTML markup. Set the [editorMode](#) property as HTML. To use HTML editing feature, inject `HtmlEditorService` in the provider section of `AppModule`.

```
`typescript
```

```
/
```

- RTE HTML Editor functionality Sample

```
*/
```

```
import { Component } from '@angular/core';
```

```
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
```

```
@Component({
```

```
selector: 'app-root',
```

```

template: `<ejs-richtexteditor #defaultRTE id='defaultRTE'>
<ng-template #valueTemplate>
<p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor
that provides the best user experience to create and update the content.
Users can format their content using standard toolbar commands.</p>
<p><b>Key features:</b></p>
<ul><li><p>Provides <IFRAME> and <DIV> modes</p></li>
<li><p>Capable of handling markdown editing.</p></li>
<li><p>Contains a modular library to load the necessary functionality on demand.</p></li>
<li><p>Provides a fully customizable toolbar.</p></li>
<li><p>Provides HTML view to edit the source directly for developers.</p></li>
<li><p>Supports third-party library integration.</p></li>
<li><p>Allows preview of modified content before saving it.</p></li>
<li><p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p></li>
<li><p>Contains undo/redo manager.</p></li>
<li><p>Creates bulleted and numbered lists.</p></li>
</ul>
</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
}
`

```

Markdown Editor

Set the [editorMode](#) property value as **Markdown** to create or edit the content and apply formatting to view markdown formatted content.

The third-party library such as **Marked** or any other library is used to convert markdown into HTML content.

- The Supported Tags are **h6,h5,h4,h3,h2,h1,blockquote,pre,p,ol,ul**.
- The Supported Selection Tags are **Bold, Italic, StrikeThrough, InlineCode, SubScript, SuperScript, UpperCase, LowerCase**.
- The supported insert commands are **Image, Link and Table**.

For further details on Markdown editing, refer to the [Markdown](#)

```
`typescript
/

    • RTE MarkDown Editor Sample

*/

import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, MarkdownEditorService } from '@syncfusion/ej2-angular-richtexteditor';

@Component({
  selector: 'app-root',
  template: `<ejs-richtexteditor #rteMarkDown id='markDown' [editorMode]='mode'>
<ng-template #valueTemplate>
```

Overview

The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor used to create, edit and return the content in valid HTML markup or markdown (MD) of the content.

The editor provides a standard toolbar to format content using its commands. Modular library features to load the necessary functionality on demand.

The toolbar contains commands to align the text, insert link, insert image, insert list, undo/redo operation, HTML view, and more.

Key features

- *Mode*: Provides IFRAME and DIV mode.
- *Module*: Modular library to load the necessary functionality on demand.
- *Toolbar*: Provide a fully customizable toolbar.
- *Editing*: HTML view to edit the source directly for developers.
- *Third-party Integration*: Supports to integrate third-party library.
- *Preview*: Preview the modified content before saving it.
- *Tools*: Handling images, hyperlinks, video, uploads and more.
- *Undo and Redo*: Undo/redo manager.
- *Lists*: Creates bulleted and numbered list.

```
</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, MarkdownEditorService]
})
export class AppComponent {
  public mode: string = 'Markdown';
}
```

[Link to the Video`](#)

Toolbar in Angular Rich text editor component

The Rich Text Editor toolbar contains a collection of tools such as bold, italic and text alignment buttons that are used to format the content. However, in most integrations, you can customize the toolbar configurations easily to suit your needs.

To get start quickly about customizing the toolbar in Angular Rich Text Editor component, refer to the video below.

To use Toolbar feature, inject `ToolbarService` in the provider section of `AppModule`.

The Rich Text Editor allows you to configure different types of toolbar using `type` field in `toolbarSettings` property. The types of toolbar are:

1. Expand
2. MultiRow

Expand Toolbar

The default mode of `toolbar` is Expand, it will hide the overflowing items in the next row. Click the expand arrow to view overflowing toolbar items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component } from '@angular/core';
import { ToolbarService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'>
<ng-template #valueTemplate>
  <p>The Rich Text Editor triggers events based on its actions. </p>
  <p>The events can be used as an extension point to perform custom
operations.</p>
  <ul>
    <li>created - Triggers when the component is rendered.</li>
    <li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
    <li>focus - Triggers when RTE is focused in.</li>
    <li>blur - Triggers when RTE is focused out.</li>
    <li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
    <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
    <li>destroyed - Triggers when the component is destroyed.</li>
  </ul>
  `
})
```

```

    </ng-template>
  </ejs-richtexteditor>`,
  providers: [ToolbarService, HtmlEditorService]
})
export class AppComponent {
  public tools: object = {
    type: 'Expand',
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Multi-row Toolbar

Set the type as **MultiRow** in [toolbarSettings](#) to hide the overflowing items in the next row. All toolbar items are visible.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component } from '@angular/core';
import { ToolbarService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'>
<ng-template #valueTemplate>
  <p>The Rich Text Editor triggers events based on its actions. </p>
  <p> The events can be used as an extension point to perform custom
operations.</p>
  <ul>
    <li>created - Triggers when the component is rendered.</li>
    <li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
    <li>focus - Triggers when RTE is focused in.</li>
    <li>blur - Triggers when RTE is focused out.</li>

```

```

        <li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
        <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
        <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, HtmlEditorService]
})
export class AppComponent {
    public tools: object = {
        type: 'MultiRow',
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Floating Toolbar

By default, the toolbar is floating at the top of the Rich Text Editor on scrolling. It can be customized by specifying the offset of the floating toolbar from document's top position using [floatingToolbarOffset](#).

Can Enable or disable the floating toolbar using [enableFloating](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, HtmlEditorService, QuickToolbarService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
import { CheckBoxComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    RichTextEditorAllModule,
    CheckBoxModule
  ],
  standalone: true,
  selector: 'app-root',

```

```

template: `<ejs-richtexteditor #typeRTE id='defaultRTE'
[toolbarSettings]='tools'>
  <ng-template #valueTemplate>
    <p>The Rich Text Editor triggers events based on its actions. </p>
    <p> The events can be used as an extension point to perform custom
operations.</p>
    <ul>
      <li>created - Triggers when the component is rendered.</li>
      <li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
      <li>focus - Triggers when RTE is focused in.</li>
      <li>blur - Triggers when RTE is focused out.</li>
      <li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
      <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
      <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
  </ng-template>
</ejs-richtexteditor>
<div>
  <ejs-checkbox #float label="Enable Floating" [checked]="true"
(change)="onChangeFloat()"></ejs-checkbox>
</div>`,
providers: [ToolbarService, HtmlEditorService, QuickToolbarService]
}))
export class AppComponent {
  @ViewChild('float') rteFloatObj: CheckBoxComponent | undefined;
  @ViewChild('typeRTE') rteObj: RichTextEditorComponent | undefined;
  public tools: object = {
    enableFloating: false
  };
  public onChangeFloat(): void {
    this.rteObj!.toolbarSettings.enableFloating =
this.rteFloatObj!.checked;
    this.rteObj!.dataBind();
  }
}

```

MAIN.TS


```














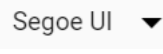
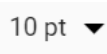


import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

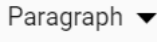
```


Toolbar Items


The following table shows that list of available tools in the Rich Text Editor's toolbar.


Name	Icons	Summary	Initialization
Undo		Allows to undo the actions.	{ items: ['Undo']}


Redo		Allows to redo the actions. toolbarSettings: { items: ['Redo']}
Alignment		Align the content with left, center, and right margin. toolbarSettings: { items: ['Alignments']}
OrderedList		Create a new list item(numbered). toolbarSettings: { items: ['OrderedList']}
UnorderedList		Create a new list item(bulleted). toolbarSettings: { items: ['UnorderedList']}
Indent		Allows to increase the indent level of the content. toolbarSettings: { items: ['Indent']}
Outdent		Allows to decrease the indent level of the content. toolbarSettings: { items: ['Outdent']}
Hyperlink		Creates a hyperlink to a text or image to a specific location in the content. toolbarSettings: { items: ['CreateLink']}
Images		Inserts an image from an online source or local computer. toolbarSettings: { items: ['Image']}
LowerCase		Change the case of selected text to lower in the content. toolbarSettings: { items: ['LowerCase']}
UpperCase		Change the case of selected text to upper in the content. toolbarSettings: { items: ['UpperCase']}
SubScript		Makes the selected text as subscript (lower). toolbarSettings: { items: ['SubScript']}
SuperScript		Makes the selected text as superscript (higher). toolbarSettings: { items: ['SuperScript']}
Print		Allows to print the editor content. toolbarSettings: { items: ['Print']}
FontName		Defines the fonts that appear under the Font Family DropDownList from the Rich Text Editor's toolbar. toolbarSettings: { items: ['FontName']}
FontSize		Defines the font sizes that appear under the Font Size DropDownList from the Rich Text Editor's toolbar. toolbarSettings: { items: ['FontSize']}
FontColor		Specifies an array of colors can be used in the colors popup for font color. toolbarSettings: { items: ['FontColor']}
BackgroundColor		Specifies an array of colors can be used in the colors popup for background color. toolbarSettings: { items: ['BackgroundColor']}


| Format |  | An Object with the options that will appear in the Paragraph Format dropdown from the toolbar. | toolbarSettings: { items: ['Formats']}|


| StrikeThrough |  | Apply double line strike through formatting for the selected text. | toolbarSettings: { items: ['StrikeThrough']}|


| ClearFormat |  | The clear format tool is useful to remove all formatting styles (such as bold, italic, underline, color, superscript, subscript, and more) from currently selected text. As a result, all the text formatting will be cleared and return to its default formatting styles. | toolbarSettings: { items: ['ClearFormat']}|


| FullScreen |  | Stretches the editor to the maximum width and height of the browser window. | toolbarSettings: { items: ['FullScreen']}|


| SourceCode |  | Rich Text Editor includes the ability for users to directly edit HTML code via “Source View”. If you made any modification in Source view directly, synchronize with Design view. | toolbarSettings: { items: ['SourceCode']}|


| NumberFormatList |  | Allows to create list items with various list style types(numbered). | toolbarSettings: { items: ['NumberFormatList']}|


| BulletFormatList |  | Allows to create list items with various list style types(bulleted). | toolbarSettings: { items: ['BulletFormatList']}|


| JustifyLeft |  | Allows each line to begin at the same distance from the editor’s left-hand side. | toolbarSettings: { items: ['JustifyLeft']} |


| JustifyCenter |  | There is an even space on each side of each line since the text is not aligned to the left or right margins. | toolbarSettings: { items: ['JustifyCenter']} |


| JustifyRight |  | Allows each line to end at the same distance from the editor’s right-hand side. | toolbarSettings: { items: ['JustifyRight']} |















| JustifyFull |  | The text is aligned with both right and left margins. | toolbarSettings: { items: ['JustifyFull']} |


| Bold |  | Text that is thicker and darker than usual. | toolbarSettings: { items: ['Bold']} |


| Italic |  | Shows a text that is leaned to the right. | toolbarSettings: { items: ['Italic']} |


| Underline |  | The underline is added to the selected text. | toolbarSettings: { items: ['Underline']} |


| ClearAll |  | Removes all styles that have been applied to the selected text. | toolbarSettings: { items: ['ClearAll']} |


- | Cut |  | Removes the text from its current location and places it into the clipboard. | toolbarSettings: { items: ['Cut']} |
- | Copy |  | The selected item is copied and pasted into the clipboard. | toolbarSettings: { items: ['Copy']} |
- | Paste |  | Allows you to insert a clipboard item into a specific location. | toolbarSettings: { items: ['Paste']} |
- | OpenLink |  | To open the URL link that is attached to the selected text. | toolbarSettings: { items: ['OpenLink']} |
- | EditLink |  | Allows you to change the URL that has been attached to a specific item. | toolbarSettings: { items: ['EditLink']} |
- | CreateTable |  | Create a table with defined columns and rows. | toolbarSettings: { items: ['CreateTable']} |
- | RemoveTable |  | Removes the selected table and its contents. | toolbarSettings: { items: ['TableRemove']} |
- | Replace |  | Replace the selected image with another image. | toolbarSettings: { items: ['Replace']} |
- | Align |  | The image can be aligned to the right, left, or center. | toolbarSettings: { items: ['Align']} |
- | Remove |  | Allows to remove the selected image from the editor. | toolbarSettings: { items: ['Remove']} |
- | OpenImageLink |  | Opens the link that is attached to the selected image. | toolbarSettings: { items: ['OpenImageLink']} |
- | EditImageLink |  | Allows to edit the link that is attached to the selected image. | toolbarSettings: { items: ['EditImageLink']} |
- | RemoveImageLink |  | Removes the link that is attached to the selected image. | toolbarSettings: { items: ['RemoveImageLink']} |
- | InsertLink |  | Allows users to add a link to a particular item. | toolbarSettings: { items: ['InsertLink']} |


| Display |  | Allows you to choose whether an image should be shown inline or as a block. |
 toolbarSettings: { items: ['Display']} |


| AltText |  | To display image description when an image on a Web page cannot be displayed. |
 toolbarSettings: { items: ['AltText']} |


| Dimension |  | Allows you to customize the image's height and width. | toolbarSettings: { items:
 ['Dimension']} |


| Maximize |  | Stretches the editor to the maximum width and height of the browser window. |
 toolbarSettings: { items: ['Maximize']} |


| Minimize |  | Shrinks the editor to the default width and height. | toolbarSettings: { items:
 ['Minimize']} |


| Preview |  | Allows to see how the editor's content looks in a browser. | toolbarSettings: { items:
 ['Preview']} |


| InsertCode |  | Represents preformatted text which is to be presented exactly as written in the
 HTML file. | toolbarSettings: { items: ['InsertCode']} |


| RemoveLink |  | Allows you to remove the applied link from the selected item. | toolbarSettings:
 { items: ['RemoveLink']} |


| TableHeader |  | Allows you to add a table header. | toolbarSettings: { items: ['TableHeader']} |

| TableColumns |  | Shows the dropdown to insert a column or delete the selected column. |
 toolbarSettings: { items: ['TableColumns']} |

| TableRows |  | Shows the dropdown to insert a row or delete the selected row. |
 toolbarSettings: { items: ['TableRows']} |

| TableCellHorizontalAlign |  | Allows the table cell content to be aligned horizontally. |
 toolbarSettings: { items: ['TableCellHorizontalAlign']} |

| TableCellVerticalAlign |  | Allows the table cell content to be aligned vertically. |
 toolbarSettings: { items: ['TableCellVerticalAlign']} |

| TableEditProperties |  | Allows you to change the table width, padding, and cell spacing styles. |
 toolbarSettings: { items: ['TableEditProperties']} |

By default, tool will be arranged in following order.

`typescript

```
items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments', 'OrderedList', 'UnorderedList', '|',
'CreateLink', 'Image', '|', 'SourceCode', 'Undo', 'Redo']
```

The tools order can be customized as our application requirement. If you are not specifying any tools order, the editor will create the toolbar with default items.

Custom tool

The Rich Text Editor allows you to configure your own commands to its toolbar using the [toolbarSettings](#) property. The command can be plain text, icon, or HTML template. The order and the group can also be defined where the command should be included. Bind action to the command by getting its instance.

This sample shows how to add your own commands to the toolbar of the Rich Text Editor. The “Ω” command is added to insert special characters in the editor. By clicking the “Ω” command, it will show the special characters list, and then choose the character to be inserted in the editor.

The following code snippet illustrates custom tool with tooltip text which will be included in [Link to the Video](#) field of the toolbarSettings property.

To get start quickly with Custom tool configuration in Angular Rich Text Editor component, refer to the video below.

In the following sample the Dialog component will be created in Rich Text Editor's `created` event. And it's target is given to Rich Text Editor's content

```
`javascript
{
  tooltipText: 'Insert Symbol',
  undo: true,
  click: this.onClick.bind(this),

  template: '<button class="e-tbar-btn e-btn" tabindex="-1" id="custom_tbar" style="width:100%"><div
class="e-tbar-btn-text" style="font-weight: 500;"> Ω</div></button>'
}
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService , RichTextEditorComponent, NodeSelection} from
 '@syncfusion/ej2-angular-richtexteditor';
import { Dialog } from '@syncfusion/ej2-popups';
@Component({
  imports: [
```

```

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor id='customRTE' #customRTE
[toolbarSettings]='tools' (created)='onCreate()'>
        <ng-template #valueTemplate>
            <div style="display:block;"><p style="margin-right:10px">The
custom
            command "insert special character" is configured as the last
item of
            the toolbar.Click on the command and choose the special
character you want
            to include from the popup.</p></div>
        </ng-template>
    </ejs-richtexteditor>
    <ejs-dialog #Dialog id="rteDialog" [buttons]='dlgButtons'
(overlayClick)="dialogOverlay()" [header]='header' [visible]='false'
[showCloseIcon]='false' [target]='target'
(created)="dialogCreate()" [isModal]='true' [cssClass]='cssClass'>
        <ng-template #content>
            <div id="rteSpecial_char">
                <div class="char_block" title="^">^</div>
                <div class="char_block" title="_">_</div>
                <div class="char_block" title="|">|</div>
                <div class="char_block" title=";">;</div>
                <div class="char_block" title="<"><</div>
                <div class="char_block" title="f">f</div>
                <div class="char_block" title="a">a</div>
                <div class="char_block" title="i">i</div>
                <div class="char_block" title="c">c</div>
                <div class="char_block" title="L">L</div>
                <div class="char_block" title="a">a</div>
                <div class="char_block" title="Y">Y</div>
                <div class="char_block" title="r">r</div>
                <div class="char_block" title="!">!</div>
                <div class="char_block" title="$">$</div>
                <div class="char_block" title="'">'</div>
                <div class="char_block" title="@">@</div>
                <div class="char_block" title="a">a</div>
                <div class="char_block" title="<"><</div>
            </div>
        </ng-template>
    </ejs-dialog>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
    @ViewChild('customRTE')
    public rteObj?: RichTextEditorComponent;
    @ViewChild('Dialog')
    public dialogObj?: Dialog;
    public selection: NodeSelection = new NodeSelection();
    public ranges?: Range;
    public tools: object = {

```

```

        items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
{
    tooltipText: 'Insert Symbol',
    undo: true,
    click: this.onClick.bind(this),
    template: '<button class="e-tbar-btn e-btn" tabindex="-1"
id="custom_tbar" style="width:100%">'
+ '<div class="e-tbar-btn-text" style="font-weight: 500;">
Ω</div></button>'
    }, '|', 'Undo', 'Redo'
    ]
};
public dlgButtons: any = [{ buttonModel: { content: "Insert", isPrimary: true
}, click: this.onInsert.bind(this) },
{ buttonModel: { content: 'Cancel' }, click: this.dialogOverlay.bind(this)
}];
public header: string = 'Special Characters';
public target: HTMLElement = document.getElementById('rteSection') as
HTMLElement;
public height: any = '350px';
public onCreate(): void {
    let customBtn: HTMLElement = document.getElementById('custom_tbar') as
HTMLElement;
    (this.dialogObj as any).target = document.getElementById('rteSection');
}
public dialogCreate(): void {
    let dialogCtn: HTMLElement = document.getElementById('rteSpecial_char')
as HTMLElement;
    dialogCtn.onclick = (e: Event) => {
        let target: HTMLElement = e.target as HTMLElement;
        let activeEle: Element =
this.dialogObj!.element.querySelector('.char_block.e-active') as Element;
        if (target.classList.contains('char_block')) {
            target.classList.add('e-active');
            if (activeEle) {
                activeEle.classList.remove('e-active');
            }
        }
    };
}
public onClick() {
    this.rteObj!.focusIn();
    this.ranges = this.selection.getRange(document);
    this.dialogObj!.width = this.rteObj!.element.offsetWidth * 0.5;
    this.dialogObj!.dataBind();
    this.dialogObj!.show();
    this.dialogObj!.element.style.maxHeight = 'none';
}
public onInsert(): void {
    let activeEle: Element =
this.dialogObj!.element.querySelector('.char_block.e-active') as Element;
    if (activeEle) {
        this.ranges!.insertNode(document.createTextNode((activeEle as
any).textContent));
    }
}

```

```

        this.dialogOverlay();
    }
    public dialogOverlay(): void {
        let activeEle: Element =
        this.dialogObj!.element.querySelector('.char_block.e-active') as Element;
        if (activeEle) {
            activeEle.classList.remove('e-active');
        }
        this.dialogObj!.hide();
    }
    public cssClass: String = "customClass e-rte-elements";
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The focus will be lost while rendering the required component for the custom toolbar, causing it to render outside the Rich Text Editor and triggering a blur event. During that time, proper functionality will not be achievable. Therefore, it is recommended to set the cssClass property or class as `e-rte-elements` in the dependency component.

Quick inline toolbar

Quick commands are opened as context-menu on clicking the corresponding element. The commands must be passed as string collection to image, text, and link attributes of the [quickToolbarSettings](#) property.

| Target Element | Default Quick Toolbar items |

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

| image | 'Replace', 'Align', 'Caption', 'Remove', 'InsertLink', 'Display', 'AltText', 'Dimension'. |

| link | 'Open', 'Edit', 'UnLink'. |

| text | null
 (Any toolbar [items](#) in the Rich Text Editor can be configured here). |

| table | 'tableHeader', 'tableRows', 'tableColumns', 'backgroundColor', '-', 'tableRemove', 'alignments', 'tableCellVerticalAlign', 'styles'. |

Custom tool can be added to the corresponding quick toolbar, using [quickToolbarSettings](#) property.

The following sample demonstrates the option to insert the image to the Rich Text Editor content as well as option to rotate the image through the quick toolbar. The image rotation functionalities have been achieved through the `toolbarClick` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditor, RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'

```



```

import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService, RichTextEditorComponent, QuickToolbarSettingsModel }
from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    ButtonModule,
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #imageRTE id='imageRTE'
[quickToolbarSettings]='quickToolbarSettings' (quickToolbarSettingsChange)=
quickToolbarSettingsChange">
    <ng-template #valueTemplate>
      <p>RichTextEditor allows to insert images from online source
as well as local
      computer where you want to insert the image in your
content.</p>
      <p><b>Get started Quick Toolbar to click on the image</b></p>
      <p>It is possible to add custom style on the selected image
inside the Rich Text Editor through quick toolbar.</p>
      
    </ng-template>
  </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
  @ViewChild('imageRTE') rteObj: RichTextEditorComponent | undefined;
  quickToolbarSettings: QuickToolbarSettingsModel = {
    image: [
      'Replace', 'Align', 'Caption', 'Remove', 'InsertLink',
'OpenImageLink', '-',
      'EditImageLink', 'RemoveImageLink', 'Display', 'AltText',
'Dimension'
    ]
  };
  quickToolbarSettingsChange: QuickToolbarSettingsModel={}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

To use quick toolbar feature, inject **QuickToolbarService, ImageService, LinkService** in the provider section of **AppModule**.

Styling in Angular Rich text editor component

Font name and Font size

By default, the editor is initialized with **Segoe UI** font-family and **10pt** font size. To change it select a different font name and font size from the drop-down in the editor's toolbar.

To apply different font style for section of the content, select the text that you would like to change, and select a required font style from the drop-down to apply the changes to the selected text.

FontName DropDowns

The following table lists the default font name and width of the fontname drop-down and available list of font names.

Default Key	Default Value
-----	-----
font name	null
width	65px
items	{ text: 'Segoe UI', value: 'Segoe UI' }, { text: 'Arial', value: 'Arial,Helvetica,sans-serif' }, { text: 'Courier New', value: 'Courier New,Courier,monospace' }, { text: 'Georgia', value: 'Georgia,serif' }, { text: 'Impact', value: 'Impact,Charcoal,sans-serif' }, { text: 'Lucida Console', value: 'Lucida Console,Monaco,monospace' }, { text: 'Tahoma', value: 'Tahoma,Geneva,sans-serif' }, { text: 'Times New Roman', value: 'Times New Roman,Times,serif' }, { text: 'Trebuchet MS', value: 'Trebuchet MS,Helvetica,sans-serif' }, { text: 'Verdana', value: 'Verdana,Geneva,sans-serif' }

FontSize DropDowns

The following table list the default font size and width of the fontsize dropdown and available list of font size.

Default Key	Default Value
-----	-----
font size	null
width	35px.
items	{ text: '8', value: '8pt' }, { text: '10', value: '10pt' }, { text: '12', value: '12pt' }, { text: '14', value: '14pt' }, { text: '18', value: '18pt' }, { text: '24', value: '24pt' }, { text: '36', value: '36pt' }.

The following sample demonstrates the option to add the font name and font size tools to the toolbar as well as modify the default width of the tools.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
```

```

imports: [
    RichTextEditorAllModule,
    DialogModule
],
standalone: true,
selector: 'app-root',
template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'
[fontSize]='size' [fontFamily] ='family'>
    </ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
    public tools: object = {
        items: [ 'FontName', 'FontSize' ]
    };
    public size = {
        width: '40px'
    };
    public family = {
        width: '60px'
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Custom fonts and size

Rich Text Editor supports to provide custom font and size with existing list.

If you want to add additional font names and font sizes to font drop-down, pass the font information as JSON data to the items field of the [fontSize](#) and [fontFamily](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
    imports: [
        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'
[fontSize]='size' [fontFamily] ='family'>
    </ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
  })
  export class AppComponent {
    public tools: object = {
      items: [ 'FontName', 'FontSize' ]
    };
    public size = {
      default: '10',
      width: '40px',
      items: [{ text: '8', value: '8pt' },
        { text: '10', value: '10pt' },
        { text: '12', value: '12pt' },
        { text: '14', value: '14pt' },
        { text: '42', value: '42pt' } ]
    };
    public family = {
      default: 'Segoe UI',
      width: '60px',
      items: [
        { text: 'Segoe UI', value: 'Segoe UI' },
        { text: 'Arial', value: 'Arial,Helvetica,sans-serif' },
        { text: 'Courier New', value: 'Courier New,Courier,monospace' },
        { text: 'Georgia', value: 'Georgia,serif' },
        { text: 'Impact', value: 'Impact,Charcoal,sans-serif' },
        { text: 'Calibri Light', value: 'CalibriLight' } ]
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Font and Background color

To apply **fontColor** or **background** color for a selected content of RTE, use font color and background color tools.

Rich Text Editor supports to provide custom font color and background color with existing list through the [colorCode](#) field of [fontColor](#) and [backgroundColor](#).

The **FontColor** and the **BackgroundColor** property has two mode of **Picker** and **Palette**. Palette mode has predefined set of colorCode. The picker mode has Color scheme to choose the color values. Through [modeSwitcher](#) you can able to switch between these two options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools' [backgroundColor]='bgColor' [fontColor] ='fontColor'>
    </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public tools: object = {
    items: [ 'FontColor', 'BackgroundColor' ]
  };
  public bgColor = {
    modeSwitcher : true
  };
  public fontColor = {
    modeSwitcher : true
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Editor content styles

By default, The content styles of Rich Text Editor are not returned while retrieving HTML value from the editor. So, the styles are not applied when using the HTML value outside of the editor. To get the styles to Rich Text Editor's content for your application, You can copy and use the below styles directly in your application. The styles listed below which used in the UI elements of the Rich Text Editor.

Make sure to add a CSS class 'e-rte-content' to the content container.

```

`css
.e-rte-content p {
margin: 0 0 10px;
margin-bottom: 10px;

```

```
}  
.e-rte-content li {  
margin-bottom: 10px;  
}  
.e-rte-content h1 {  
font-size: 2.17em;  
font-weight: 400;  
line-height: 1;  
margin: 10px 0;  
}  
.e-rte-content h2 {  
font-size: 1.74em;  
font-weight: 400;  
margin: 10px 0;  
}  
.e-rte-content h3 {  
font-size: 1.31em;  
font-weight: 400;  
margin: 10px 0;  
}  
.e-rte-content h4 {  
font-size: 1em;  
font-weight: 400;  
margin: 0;  
}  
.e-rte-content h5 {  
font-size: 0.8em;  
font-weight: 400;  
margin: 0;  
}  
.e-rte-content h6 {  
font-size: 0.65em;  
font-weight: 400;
```

```
margin: 0;
}
.e-rte-content blockquote {
margin: 10px 0;
margin-left: 0;
padding-left: 5px;
}
.e-rte-content pre {
background-color: inherit;
border: 0;
border-radius: 0;
color: #333;
font-size: inherit;
line-height: inherit;
margin: 0 0 10px;
overflow: visible;
padding: 0;
white-space: pre-wrap;
word-break: inherit;
word-wrap: break-word;
}
.e-rte-content strong, .e-rte-content b {
font-weight: 700;
}
.e-rte-content a {
text-decoration: none;
-webkit-user-select: auto;
-ms-user-select: auto;
user-select: auto;
}
.e-rte-content a:hover {
text-decoration: underline;
}
```

```
.e-rte-content h3 + h4,  
.e-rte-content h4 + h5,  
.e-rte-content h5 + h6 {  
margin-top: 00.6em;  
}  
  
.e-rte-content .e-rte-image.e-imgbreak {  
border: 0;  
cursor: pointer;  
display: block;  
float: none;  
margin: 5px auto;  
max-width: 100%;  
position: relative;  
}  
  
.e-rte-content .e-rte-image {  
border: 0;  
cursor: pointer;  
display: block;  
float: none;  
margin: auto;  
max-width: 100%;  
position: relative;  
}  
  
.e-rte-content .e-rte-image.e-imginline {  
display: inline-block;  
float: none;  
margin-left: 5px;  
margin-right: 5px;  
max-width: calc(100% - (2 * 5px));  
vertical-align: bottom;  
}  
  
.e-rte-content .e-rte-image.e-imgcenter {  
cursor: pointer;
```



```
display: block;
float: none;
margin: 5px auto;
max-width: 100%;
position: relative;
}

.e-rte-content .e-rte-image.e-imgleft {
float: left;
margin: 0 5px 0 0;
text-align: left;
}

.e-rte-content .e-rte-image.e-imgright {
float: right;
margin: 0 0 0 5px;
text-align: right;
}

.e-rte-content .e-rte-img-caption {
display: inline-block;
margin: 5px auto;
max-width: 100%;
position: relative;
}

.e-rte-content .e-rte-img-caption.e-caption-inline {
display: inline-block;
margin: 5px auto;
margin-left: 5px;
margin-right: 5px;
max-width: calc(100% - (2 * 5px));
position: relative;
text-align: center;
vertical-align: bottom;
}

.e-rte-content .e-rte-img-caption.e-imgcenter {
```

```
display: block;
}
.e-rte-content .e-rte-img-caption .e-rte-image.e-imgright,
.e-rte-content .e-rte-img-caption .e-rte-image.e-ingleft {
float: none;
margin: 0;
}
.e-rte-content .e-rte-table {
border-collapse: collapse;
empty-cells: show;
}
.e-rte-content .e-rte-table td,
.e-rte-content .e-rte-table th {
border: 1px solid #bdbdbd;
height: 20px;
min-width: 20px;
padding: 2px 5px;
vertical-align: middle;
}
.e-rte-content .e-rte-table.e-dashed-border td,
.e-rte-content .e-rte-table.e-dashed-border th {
border-style: dashed;
}
.e-rte-content .e-rte-img-caption .e-img-inner {
box-sizing: border-box;
display: block;
font-size: 16px;
font-weight: initial;
margin: auto;
opacity: .9;
position: relative;
text-align: center;
width: 100%;
```

```

}
.e-rte-content .e-rte-img-caption .e-img-wrap {
display: inline-block;
margin: auto;
padding: 0;
width: 100%;
}
.e-rte-content blockquote {
border-left: solid 2px #333;
}
.e-rte-content a {
color: #2e2ef1;
}
.e-rte-content .e-rte-table th {
background-color: #e0e0e0;
}
`

```

Image in Angular Rich text editor component

Rich Text Editor allows to insert images in your content from online sources as well as local computer. For inserting an image to the Rich Text Editor, the following list of options have been provided in the [insertImageSettings](#)

Options	Description
----- -----	
allowedTypes	Specifies the extensions of the image types allowed to insert on bowering and passing the extensions with comma separators. For example, pass allowedTypes as .jpg and .png.
display	Sets the default display for an image when it is inserted in to the Rich Text Editor. Possible options are: 'inline' and 'block'.
width	Sets the default width of the image when it is inserted in the Rich Text Editor.
height	Sets the default height of the image when it is inserted in the Rich Text Editor.
saveUrl	Provides URL to map the action result method to save the image.
path	Specifies the location to store the image.
resize	To enable resizing for image element.
minWidth	Defines the maximum Width of the image.
maxWidth	Defines the maximum Width of the image.
minHeight	Defines the minimum Height of the image.

| [maxHeight](#) | Defines the maximum Height of the image. |

| [resizeByPercent](#) | Image resizing should be done by percentage calculation. |

Upload options

Through the **browse** option in the Image dialog, select the image from the local machine and insert into the Rich Text Editor content.

If the path field is not specified in the [insertImageSettings](#), the image will be transferred into base 64 and blob url for the image will be created and the generated url will be set to the src property of img tag.

`typescript

```

```

If you want to insert a lot of tiny images in the editor and don't want a specific physical location for saving images, you can opt to save format as Base64.

In the following sample, the image has been loaded from the local machine and it will be saved in the given location.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='iframeRTE' [toolbarSettings]='tools'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor
        that provides the best user experience to create and update the content.
        Users can format their content using standard toolbar commands.</p>
      <p><b>Key features:</b></p>
      <ul><li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;;
        modes</p></li>
        <li><p>Capable of handling markdown editing.</p></li>
        <li><p>Contains a modular library to load the necessary functionality on demand.</p></li>
        <li><p>Provides a fully customizable toolbar.</p></li>
        <li><p>Provides HTML view to edit the source directly for developers.</p></li>
```

```

        <li><p>Supports third-party library integration.</p></li>
        <li><p>Allows preview of modified content before saving
it.</p></li>
        <li><p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p></li>
        <li><p>Contains undo/redo manager.</p></li>
        <li><p>Creates bulleted and numbered lists.</p></li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
    public tools: object = {
        items: ['Image']
    };
}

```

MAIN.TS

```

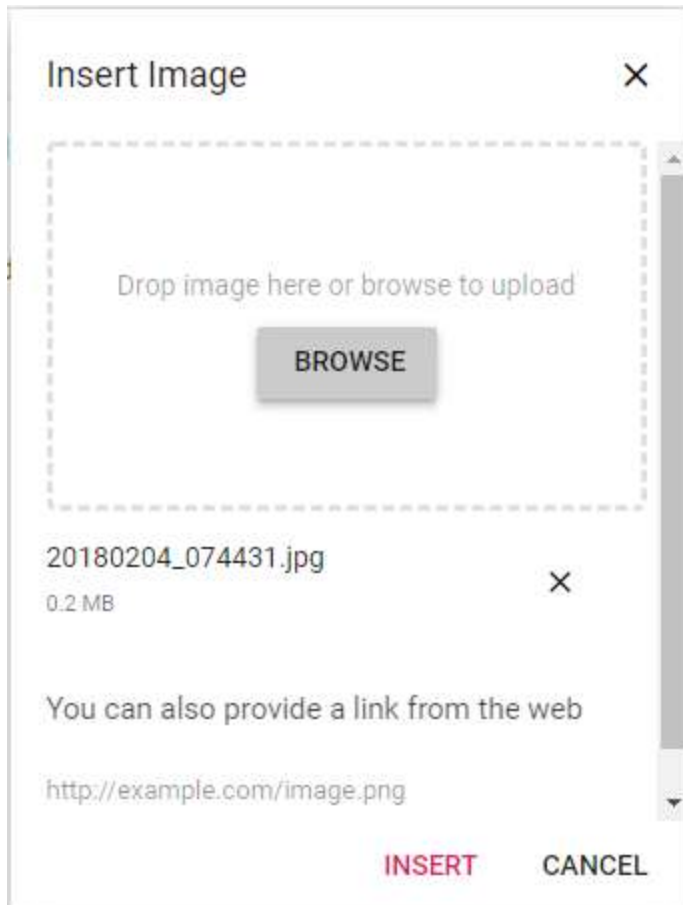
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Delete Image

To remove an image from the Rich Text Editor content, select the image and click Remove tool from the quick toolbar. It will delete the image from the RTE content as well as from the service location if the `removeUrl` is given.

Once you select the image from the local machine, the URL for the image will be generate. From there,you can remove the image from the service location by clicking the cross icon.



The following sample explains, how to configure `removeUrl` to remove a saved image from the remote service location, when the following image remove actions are performed:

- `delete` key action.
- `backspace` key action.
- Removing uploaded image file from the insert image dialog.
- Deleting image using the quick toolbar `remove` option.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
/**
 * RTE - removeUrl Sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
```

```

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor [toolbarSettings]='toolbarSettings'
[insertImageSettings]='insertImageSettings'>
        <ng-template #valueTemplate>
            <p>The Rich Text Editor is WYSIWYG ("what you see is what you
get") editor useful to create and edit content, and return the valid <a
href="https://ej2.syncfusion.com/home/" target="_blank">HTML markup</a> or <a
href="https://ej2.syncfusion.com/home/" target="_blank">markdown</a> of the
content</p>
            <p><b>Key features:</b></p>
            <ul>
                <li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;;
modes</p></li>
                <li><p>Capable of handling markdown editing.</p></li>
                <li><p>Contains a modular library to load the necessary
functionality on demand.</p></li>
                <li><p>Provides a fully customizable toolbar.</p></li>
                <li><p>Provides HTML view to edit the source directly for
developers.</p></li>
                <li><p>Supports third-party library integration.</p></li>
                <li><p>Allows preview of modified content before saving
it.</p></li>
                <li><p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p></li>
            </ul>
        </ng-template>
    </ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
    public toolbarSettings: Object = {
        items: ['Image']
    };
    public insertImageSettings: Object = {
        saveUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Save',
        removeUrl: 'https://ej2.syncfusion.com/services/api/uploadbox/Remove'
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

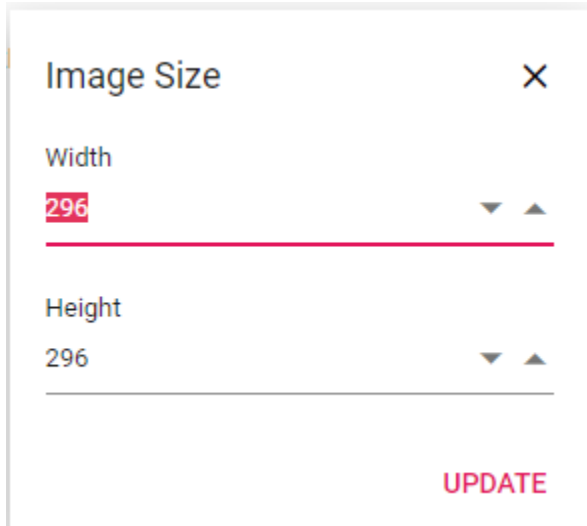
Insert from web

To insert an image from the online source like Google, Ping, etc., you should enable the image tool on the editor's toolbar. By default, the image tool opens a simple dialog which allows you to insert an image from online source.

Dimension

Sets the default width and height of the image when it is inserted in the Rich Text Editor using [width](#) and [height](#) of the [insertImageSettings](#) property.

Through the quick toolbar, change the width and height using **Change Size** option. Once you click, the Image Size dialog box will open as follows. In that you can specify the width and height of the image in pixel.



Caption and Alt Text

Image caption and alternative text can be specified for the inserted image in the Rich Text Editor through the [quickToolbarSettings](#) property. It has following two options,

- Image Caption
- Alternative Text.

Through the Alternative Text option, set the alternative text for the image, when the image is not upload successfully into the Rich Text Editor.

By clicking the Image Caption, the image will get wrapped in an image element with a caption. Then, you can type caption content inside the Rich Text Editor.

Display position

Sets the default display for an image when it is inserted in the Rich Text Editor using [display](#) field in [insertImageSettings](#). It has two possible options: 'inline' and 'block'.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, QuickToolbarService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
```



```

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor #imageRTE id='imageRTE'
[insertImageSettings]='insertImageSettings'>
        <ng-template #valueTemplate>
            <p>Rich Text Editor allows to insert images from online source
as well as local
            computer where you want to insert the image in your
content.</p>
            <p><b>Get started Quick Toolbar to click on the image</b></p>
            <p>It is possible to add custom style on the selected image
inside the Rich Text Editor through quick toolbar.</p>
            
        </ng-template>
    </ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
    public insertImageSettings = {
        display: 'inline'
    };
}

```

MAIN.TS

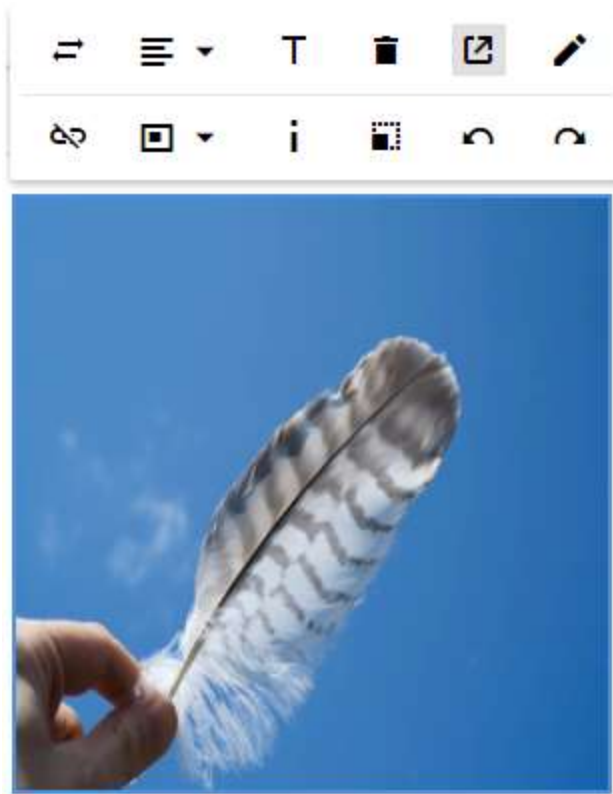
```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Image with link

The hyperlink itself can be an image in Rich Text Editor. If the image given as hyperlink, remove, edit and open link will be added to the quick toolbar of image. For further details about link, see the [link documentation](#) documentation.



Resize

Rich Text Editor has a built-in image inserting support. The resize points will be appearing on each corner of image when focus. So, users can resize the image using mouse points or thumb through the resize points easily. Also, the resize calculation will be done based on aspect ratio.



Drag and Drop

By default, the Rich Text Editor allows you to insert images by drag-and-drop from the local file system such as Windows Explorer into the content editor area. And, you can upload the images to the server before inserting into the editor by configuring the saveUrl property. The images can be repositioned anywhere within the editor area by dragging and dropping the image.

In the following sample, you can see feature demo.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, QuickToolbarService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor
[insertImageSettings]='insertImageSettings'>
  <ng-template #valueTemplate>
    <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content. Users can format their content using standard toolbar
commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;
modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary
functionality on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving
it.</p>
      </li>
      <li>
        <p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p>
      </li>
      <li>
        <p>Contains undo/redo manager.</p>
      </li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`
})
export class AppComponent {}

```

```

        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
    public insertImageSettings = {
        saveUrl :
'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save'
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Drag and drop with specific extension images

You can allow the specific images alone to be uploaded using the the allowedTypes property. By default, the Rich Text Editor allows the JPG, JPEG, and PNG formats. You can configure this formats as follows.

`typescript

```

insertImageSettings: {
    allowedTypes: ['.jpg']
}

```

`

Prevent drag and drop action

You can prevent drag-and-drop action by setting the actionBegin argument cancel value to true. The following code shows how to prevent the drag-and-drop.

`typescript

```

actionBegin: function (args: any): void {
    if(args.type === 'drop' || args.type === 'dragstart') {
        args.cancel =true;
    }
}

```

`

Audio in Angular Rich text editor component

The Rich Text Editor allows you to insert audio from online sources and local computers and then insert them into your content. You can insert the audio with the following list of options in the [insertAudioSettings](#) property.

Options	Description
allowedTypes	Specifies the extensions of the audio types allowed to insert on bowering and passing the extensions with comma separators. For example, pass allowedTypes as <code>.mp3</code> , <code>.wav</code> , <code>.m4a</code> and <code>.wma</code> .
layoutOption	Sets the default display for audio when it is inserted into the Rich Text Editor. Possible options are <code>Inline</code> and <code>Break</code> .
saveFormat	Sets the default save format of the audio element when inserted. Possible options are: <code>Blob</code> and <code>Base64</code> .
saveUrl	Provides URL to map the action result method to save the audio.
removeUrl	Provides URL to map the action result method to remove the audio.
path	Specifies the location to store the audio.

Configure audio tool in the toolbar

You can add an `audio` tool in the Rich Text Editor toolbar using the `toolbarSettings.items` property.

Rich Text Editor features are segregated into individual feature-wise modules. In this demo, we have used the following injectable services `ToolbarService`, `LinkService`, `HtmlEditorService`, `AudioService` in the `@NgModule.providers` section.

To configure the `Audio` toolbar item, refer to the below code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService,
AudioService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='' [toolbarSettings]='toolbarSettings'>
<ng-template #valueTemplate>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what
you get") editor`
})
```

that provides the best user experience to create and update the content.

Users can format their content **using standard** toolbar commands.</p>
 <p>Key features:</p>

 <p>Provides <IFRAME> and <DIV> modes</p>

 <p>Capable of handling markdown editing.</p>

 <p>Contains a modular library to load the necessary functionality on demand.</p>

 <p>Provides a fully customizable toolbar.</p>

 <p>Provides HTML view to edit the source directly **for** developers.</p>

 <p>Supports third-party library integration.</p>

 <p>Allows preview of modified content before saving it.</p>

 <p>Handles images, hyperlinks, video, hyperlinks, uploads, etc.</p>

 <p>Contains undo/redo manager.</p>

 <p>Creates bulleted and numbered lists.</p>

 </ng-template>
 </ejs-richtexteditor>`,
 providers: [
 ToolbarService, QuickToolbarService, LinkService, AudioService,
 HtmlEditorService,
],
})
export class AppComponent {
 public toolbarSettings: object = {
 items: ['Audio']
 };
}

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
```

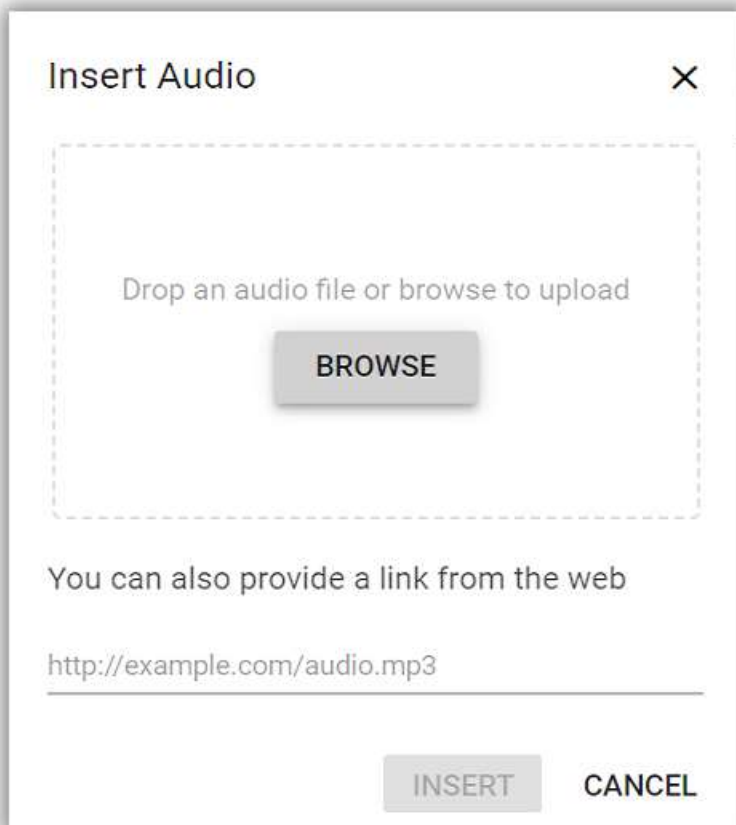
```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Insert audio from the web

You can insert audio from either the hosted link or the local machine, by clicking the audio button in the editor's toolbar. On clicking the audio button, a dialog opens, which allows you to insert audio from the web URL.

Insert from web URL

By default, the audio tool opens the audio dialog, allowing you to insert audio from an online source. Inserting the URL will be added to the `src` attribute of the `<source>` tag.



Insert audio from local machine

You can use the `browse` option on the audio dialog, to select the audio from the local machine and insert it into the Rich Text Editor content.

If the path field is not specified in the [insertAudioSettings](#), the audio will be converted into the `Blob` URL or `Base64` and inserted inside the Rich Text Editor.

Restrict audio upload based on size

You can restrict the audio uploaded from the local machine when the uploaded audio file size is greater than the allowed size by using the [fileUploading](#) event.

The file size in the argument will be returned in `bytes`.

In the following illustration, the audio size has been validated before uploading, and it is determined whether the audio has been uploaded or not.

```
`typescript
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, AudioService } from
 '@syncfusion/ej2-angular-richtexteditor';

import { UploadingEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component( {
  selector: 'app-root',

  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertAudioSettings] =
 'insertAudioSettings' (fileUploading) = 'onAudioUpload($event)'" >
</ejs-richtexteditor>`,

  providers: [
    ToolbarService, QuickToolbarService, LinkService, AudioService, HtmlEditorService,
  ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: [ 'Audio' ]
  };

  public insertAudioSettings: Object = {
    saveUrl: 'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save',
    path: '../Files/'
  };

  public onAudioUpload = ( args: UploadingEventArgs ) => {
    let sizeInBytes: number = args.fileData.size;
    let fileSize: number = 500000;
    if ( fileSize < sizeInBytes ) {
      args.cancel = true;
    }
  }
}
```


Server-side action

The selected audio can be uploaded to the required destination using the controller action below. Map this method name in [insertAudioSettings.saveUrl](#) and provide the required destination path through [insertAudioSettings.path](#) properties.

If you want to insert lower-sized audio files in the editor and don't want a specific physical location for saving the audio, you can opt to save the format as **Base64**.

In the following code blocks, the audio module has been injected and can insert the audio files saved in the specified path.

```
`typescript
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, AudioService } from
'@syncfusion/ej2-angular-richtexteditor';

@Component( {
  selector: 'app-root',
  template: '<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertAudioSettings] =
'insertAudioSettings' >
</ejs-richtexteditor>',
  providers: [
    ToolbarService, QuickToolbarService, LinkService, AudioService, HtmlEditorService,
  ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: [ 'Audio' ]
  };
  public insertAudioSettings: Object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  };
}
`

`csharp
using System;
using System.IO;
using FileUpload.Models;
```

```
using System.Diagnostics;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using System.Collections.Generic;
using Microsoft.AspNetCore.Hosting;
namespace FileUpload.Controllers
{
    public class HomeController : Controller
    {
        private IHostingEnvironment hostingEnv;
        public HomeController(IHostingEnvironment env)
        {
            hostingEnv = env;
        }
        public IActionResult Index()
        {
            return View();
        }
        [AcceptVerbs("Post")]
        public void SaveFiles(IList<IFormFile> UploadFiles)
        {
            try
            {
                foreach (IFormFile file in UploadFiles)
                {
                    if (UploadFiles != null)
                    {
                        string filename = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim('"');
                        filename = hostingEnv.WebRootPath + "\\Files" + $"@\"{filename}";
                        // Create a new directory, if it does not exists
                        if (!Directory.Exists(hostingEnv.WebRootPath + "\\Files"))
                        {
```

```

Directory.CreateDirectory(hostingEnv.WebRootPath + "\\Files");
}
if (!System.IO.File.Exists(filename))
{
    using (FileStream fs = System.IO.File.Create(filename))
    {
        file.CopyTo(fs);
        fs.Flush();
    }
    Response.StatusCode = 200;
}
}
}
}
catch (Exception)
{
    Response.StatusCode = 204;
}
}
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}
}
}
,

```

Audio save format

The audio files can be saved as **Blob** or **Base64** URL by using the [insertAudioSettings.saveFormat](#) property, which is of enum type, and the generated URL will be set to the **src** attribute of the **<source>** tag.

The default **saveFormat** property is set to **Blob** format.

```

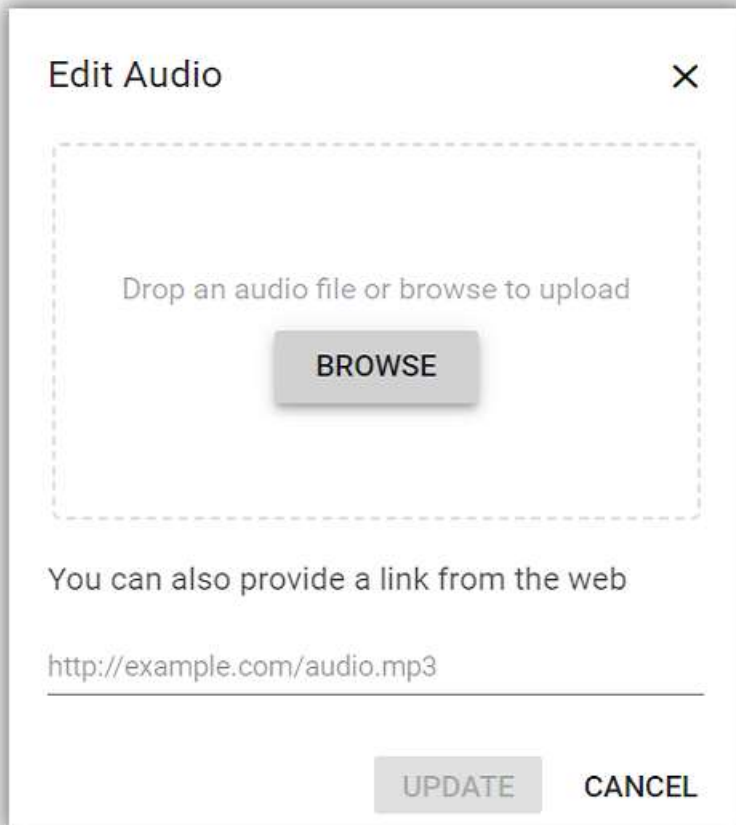
`HTML
<audio>

```

```
<source src="blob:http://ej2.syncfusion.com/3ab56a6e-ec0d-490f-85a5-f0aeb0ad8879"
type="audio/mp3" >
</audio>
<audio>
<source src="data:audio/mp3;base64,iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAYAAABXAvmHA"
type="audio/mp3" >
</audio>
`
```

Replacing audio

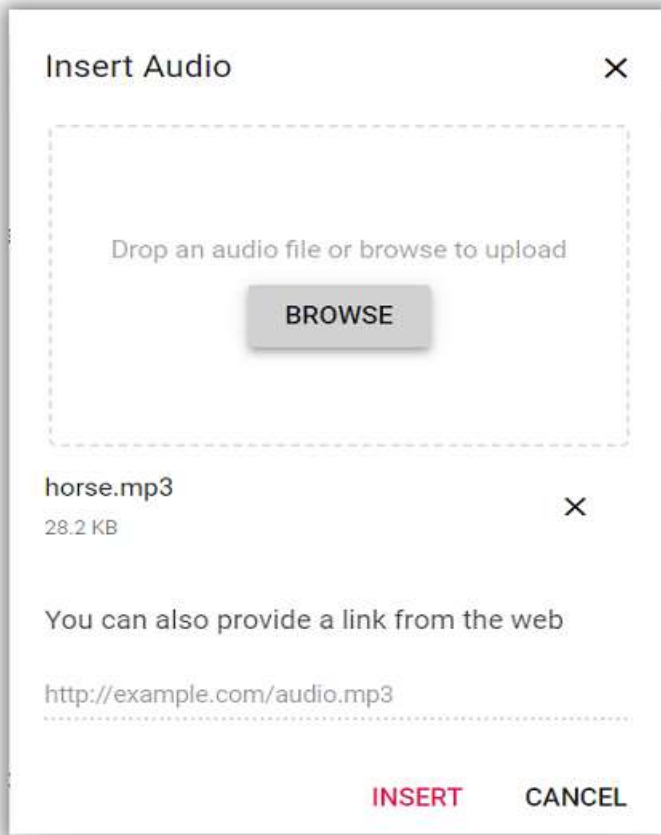
Once an audio file has been inserted, you can change it using the Rich Text Editor [quickToolbarSettings](#) `audioReplace` option. You can replace the audio file using the web URL or the browse option in the audio dialog.



Delete audio

To remove audio from the Rich Text Editor content, select the audio and click the `audioRemove` button from the quick toolbar. It will delete the audio from the Rich Text Editor content as well as from the service location if the [insertAudioSettings.removeUrl](#) is given.

Once you select the audio from the local machine, the URL for the audio will be generated. You can remove the audio from the service location by clicking the cross icon.



Display position

Sets the default display property for audio when it is inserted in the Rich Text Editor using the [insertAudioSettings.layoutOption](#) property. It has two possible options: **Inline** and **Break**. When updating the display positions, it updates the audio elements' layout position.

The default **layoutOption** property is set to **Inline**.

``typescript`

```
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, AudioService } from
'@syncfusion/ej2-angular-richtexteditor';

@Component( {
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertAudioSettings] =
'insertAudioSettings' >
</ejs-richtexteditor>`,
  providers: [ ToolbarService, QuickToolbarService, LinkService, AudioService, HtmlEditorService, ],
})
export class AppComponent {
```

```
public toolbarSettings: object = {
  items: [ 'Audio' ],
};
public insertAudioSettings: Object = {
  layoutOption: 'Inline'
};
}
```

Rename audio before inserting

You can use the [insertAudioSettings](#) property, to specify the server handler to upload the selected audio. Then by binding the [fileUploadSuccess](#) event, you can receive the modified file name from the server and update it in the Rich Text Editor's insert audio dialog.

`HTML

```
<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertAudioSettings] = 'insertAudioSettings'
(fileUploadSuccess) = 'onAudioUploadSuccess($event)' >
```

```
<ng-template #valueTemplate>
```

```
<p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
content</p>
```

```
</ng-template>
```

```
</ejs-richtexteditor>
```

`typescript

```
import { Component } from '@angular/core';
```

```
import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, AudioService, } from
 '@syncfusion/ej2-angular-richtexteditor';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: 'app.component.html',
```

```
  providers: [ ToolbarService, QuickToolbarService, LinkService, AudioService, HtmlEditorService ],
```

```
})
```

```
export class AppComponent {
```

```
  public toolbarSettings: object = {
```

```
    items: ['Audio'],
```

```
  };
```

```

public insertAudioSettings: Object = {
  saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
  path: "[SERVICEHOSTEDPATH]/Files/"
};

public onAudioUploadSuccess = (args: any) => {
  alert("Get the new file name here");
  if( args.e.currentTarget.getResponseHeader('name') != null ){
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let fileName : any = document.querySelector(".e-file-name")[0];
    fileName.innerHTML = args.fileData.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML, "");
    fileName.title = args.fileData.name;
  }
};
}
`

```

To configure the server-side handler, refer to the below code.

```

`csharp
int x = 0;
string file;
[AcceptVerbs("Post")]
public void Rename()
{
  try
  {
    var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
    fileName = httpPostedFile.FileName;
    if (httpPostedFile != null)
    {
      var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");
      if (!Directory.Exists(fileSave))
      {
        Directory.CreateDirectory(fileSave);

```

```

}
var fileName = Path.GetFileName(httpPostedFile.FileName);
var fileSavePath = Path.Combine(fileSave, fileName);
while (System.IO.File.Exists(fileSavePath))
{
    fileName = "rteFiles" + x + "-" + fileName;
    fileSavePath = Path.Combine(fileSave, fileName);
    x++;
}
if (!System.IO.File.Exists(fileSavePath))
{
    httpPostedFile.SaveAs(fileSavePath);
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.Headers.Add("name", fileName);
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusDescription = "File uploaded succesfully";
    Response.End();
}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
}
,

```


Upload audio with authentication

You can add additional data with the audio uploaded from the Rich Text Editor on the client side, which can even be received on the server side by using the [fileUploading](#) event and its `customFormData` argument, you can pass parameters to the controller action. On the server side, you can fetch the custom headers by accessing the form collection from the current request, which retrieves the values sent using the POST method.

By default, it doesn't support the `UseDefaultCredentials` property; we need to manually append the default credentials with the upload request.

```
`typescript
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, AudioService, } from
 '@syncfusion/ej2-angular-richtexteditor';

import { UploadingEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component({
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertAudioSettings] =
 'insertAudioSettings' (fileUploading) = 'onAudioUpload($event)' >
</ejs-richtexteditor>`,
  providers: [ ToolbarService, QuickToolbarService, LinkService, AudioService, HtmlEditorService ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: ['Audio'],
  };
  public insertAudioSettings: Object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  };
  public onAudioUpload = (args: UploadingEventArgs) => {
    let accessToken = "Authorization_token";
    // adding custom form Data
    args.customFormData = [ { 'Authorization': accessToken }];
  };
}
```

```
`csharp
public void SaveFiles(IList<IFormFile> UploadFiles)
{
    string currentPath = Request.Form["Authorization"].ToString();
}
`
```

See Also

- [How to edit the quick toolbar settings](#)
- [How to use the link editing option in the toolbar items](#)

Video in Angular Rich text editor component

The Rich Text Editor allows you to insert videos from online sources and local computers and then insert them into your content. You can insert the video with the following list of options in the [insertVideoSettings](#) property.

| Options | Description |

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

| [allowedTypes](#) | Specifies the extensions of the video types allowed to insert on bowering and passing the extensions with comma separators. For example, pass allowedTypes as `.mp4`, `.mov`, `.wmv` and `.avi`.|

| [layoutOption](#) | Sets the default display for a video when it is inserted into the Rich Text Editor. Possible options are: `Inline` and `Break`.|

| [saveFormat](#) | Sets the default save format of the video element when inserted. Possible options are: `Blob` and `Base64`.|

| [width](#) | Sets the default width of the video when it is inserted in the Rich Text Editor.|

| [minWidth](#) | Sets the minWidth of the video element when it is inserted in the Rich Text Editor.|

| [maxWidth](#) | Sets the maxWidth of the video element when it is inserted in the Rich Text Editor.|

| [height](#) | Sets the default height of the video when it is inserted in the Rich Text Editor.|

| [minHeight](#) | Sets the minHeight of the video element when it is inserted in the Rich Text Editor.|

| [maxHeight](#) | Sets the maxHeight of the video element when it is inserted in the Rich Text Editor.|

| [saveUrl](#) | Provides URL to map the action result method to save the video.|

| [removeUrl](#) | Provides URL to map the action result method to remove the video.|

| [path](#) | Specifies the location to store the video.|

| [resize](#) | Sets the resizing action for the video element.|

| [resizeByPercent](#) | Sets the percentage values for the video element with the resizing action.|

Configure the video tool in the toolbar

You can add the `video` tool in the Rich Text Editor toolbar using the `toolbarSettings` [items](#) property.

Rich Text Editor features are segregated into individual feature-wise modules. In this demo, we have used the following injectable services `ToolbarService`, `LinkService`, `HtmlEditorService`, `VideoService` in the `@NgModule.providers` section.

To configure the `Video` toolbar item, refer to the below code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='' [toolbarSettings]='toolbarSettings'>
<ng-template #valueTemplate>
  <p>The Rich Text Editor component is WYSIWYG ("what you see is what you get") editor
    that provides the best user experience to create and update the
    content.
    Users can format their content using standard toolbar
    commands.</p>
    <p><b>Key features:</b></p>
    <ul>
      <li>
        <p>Provides &#60;IFRAME&#62; and &#60;DIV&#62; modes</p>
      </li>
      <li>
        <p>Capable of handling markdown editing.</p>
      </li>
      <li>
        <p>Contains a modular library to load the necessary
        functionality on demand.</p>
      </li>
      <li>
        <p>Provides a fully customizable toolbar.</p>
      </li>
      <li>
        <p>Provides HTML view to edit the source directly for
        developers.</p>
      </li>
      <li>
        <p>Supports third-party library integration.</p>
      </li>
      <li>
        <p>Allows preview of modified content before saving it.</p>
      </li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`
})
```

```

        <li>
            <p>Handles images, hyperlinks, video, hyperlinks, uploads,
etc.</p>
        </li>
        <li>
            <p>Contains undo/redo manager.</p>
        </li>
        <li>
            <p>Creates bulleted and numbered lists.</p>
        </li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
    providers: [
        ToolbarService, QuickToolbarService, LinkService, VideoService,
        HtmlEditorService,
    ],
} )
export class AppComponent {
    public toolbarSettings: object = {
        items: [ 'Video' ]
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

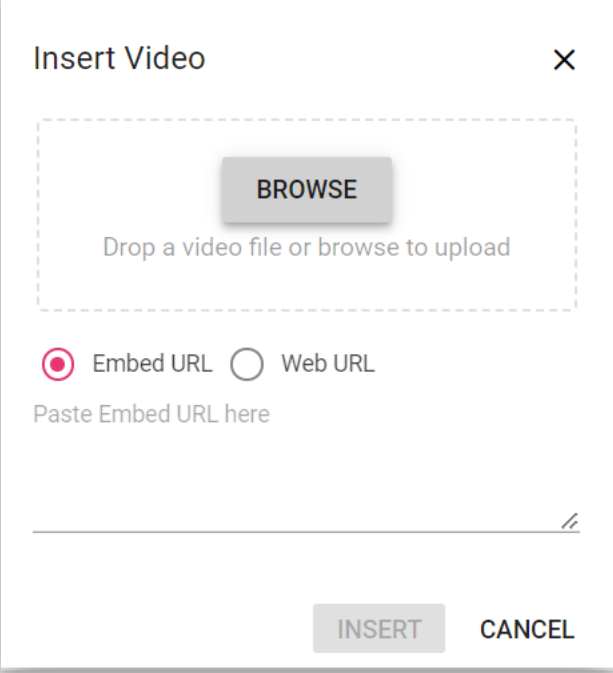
```

Insert a video from the web

You can insert a video from either the hosted link or the local machine by clicking the video button in the editor's toolbar. On Clicking the Video button, a dialog opens which allows you to insert video from the Embedded URL or web URL.

Insert from embed URL

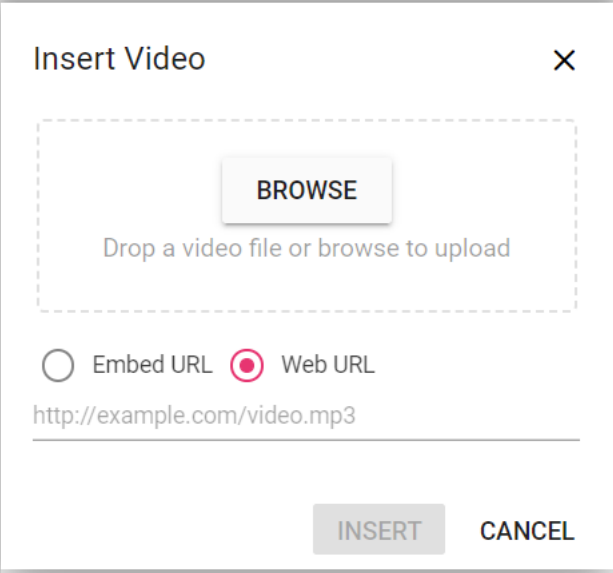
The insert video dialog opens with the **Embed URL** option as default which allows you to insert an embedded URL.



The dialog box is titled "Insert Video" with a close button (X) in the top right corner. It features a dashed rectangular area for file dropping, containing a "BROWSE" button and the text "Drop a video file or browse to upload". Below this, there are two radio buttons: "Embed URL" (which is selected, indicated by a red dot) and "Web URL". Under the "Embed URL" option, there is a text input field with the placeholder text "Paste Embed URL here". At the bottom of the dialog, there are two buttons: "INSERT" and "CANCEL".

Insert from web URL

You can switch to **Web URL** by selecting the web URL check box. Inserting with the web URL option will add the video URL as the `src` attribute of the `<source>` tag.



This dialog box is identical in layout to the first one, but the "Web URL" radio button is selected instead of "Embed URL". The text input field below the radio buttons now contains the example URL "http://example.com/video.mp3". The "BROWSE" button and the "Drop a video file or browse to upload" text are still present in the dashed area at the top.

Insert video from local machine

You can use the **browse** option on the video dialog, to select the video from the local machine and insert it into the Rich Text Editor content.

If the path field is not specified in the [insertVideoSettings](#), the video will be converted into the **Blob** URL or **Base64** and inserted inside the Rich Text Editor.

Restrict video upload based on size

You can restrict the video uploaded from the local machine when the uploaded video file size is greater than the allowed size by using the [fileUploading](#) event.

The file size in the argument will be returned in **bytes**.

In the following example, the video size has been validated before uploading and determined whether the video has been uploaded or not.

```
`typescript
import { Component } from '@angular/core';
import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService } from
 '@syncfusion/ej2-angular-richtexteditor';
import { UploadingEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component( {
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertVideoSettings] =
  'insertVideoSettings' (fileUploading) = 'onVideoUpload($event)' >
  </ejs-richtexteditor>`,
  providers: [
    ToolbarService, QuickToolbarService, LinkService, VideoService, HtmlEditorService,
  ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: [ 'Video' ]
  };
  public insertVideoSettings: Object = {
    saveUrl: 'https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save',
    path: '../Files/'
  };
  public onVideoUpload = ( args: UploadingEventArgs ) => {
    let sizeInBytes: number = args.fileData.size;
    let fileSize: number = 500000;
    if ( fileSize < sizeInBytes ) {
      args.cancel = true;
    }
  }
}
```

```
}
,
```

Server-side action

The selected video can be uploaded to the required destination using the controller action below. Map this method name in [insertVideoSettings.saveUrl](#) and provide the required destination path through [insertVideoSettings.path](#) properties.

If you want to insert lower-sized video files in the editor and don't want a specific physical location for saving the video, you can save the format as `Base64`.

In the following code blocks, the video module has been injected and can insert the video files saved in the specified path.

```
`typescript
```

```
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService } from
 '@syncfusion/ej2-angular-richtexteditor';
```

```
@Component( {
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertVideoSettings] =
'insertVideoSettings' >`
  </ejs-richtexteditor>`,
```

```
  providers: [
    ToolbarService, QuickToolbarService, LinkService, VideoService, HtmlEditorService,
  ],
})
```

```
export class AppComponent {
  public toolbarSettings: object = {
    items: [ 'Video' ]
  };
  public insertVideoSettings: Object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  };
}
```

```
,
```

```
`csharp
```

```
using System;
```

```
using System.IO;
using FileUpload.Models;
using System.Diagnostics;
using System.Net.Http.Headers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Http;
using System.Collections.Generic;
using Microsoft.AspNetCore.Hosting;
namespace FileUpload.Controllers
{
    public class HomeController : Controller
    {
        private IHostingEnvironment hostingEnv;
        public HomeController(IHostingEnvironment env)
        {
            hostingEnv = env;
        }
        public IActionResult Index()
        {
            return View();
        }
        [AcceptVerbs("Post")]
        public void SaveFiles(IList<IFormFile> UploadFiles)
        {
            try
            {
                foreach (IFormFile file in UploadFiles)
                {
                    if (UploadFiles != null)
                    {
                        string filename = ContentDispositionHeaderValue.Parse(file.ContentDisposition).FileName.Trim("");
                        filename = hostingEnv.WebRootPath + "\\Files" + $"@\"{filename}";
                        // Create a new directory, if it does not exists
                    }
                }
            }
        }
    }
}
```



```

if (!Directory.Exists(hostingEnv.WebRootPath + "\\Files"))
{
    Directory.CreateDirectory(hostingEnv.WebRootPath + "\\Files");
}
if (!System.IO.File.Exists(filename))
{
    using (FileStream fs = System.IO.File.Create(filename))
    {
        file.CopyTo(fs);
        fs.Flush();
    }
    Response.StatusCode = 200;
}
}
}
}
catch (Exception)
{
    Response.StatusCode = 204;
}
}
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
}
}
,

```

Video save format

The video files can be saved as **Blob** or **Base64** URL by using the [insertVideoSettings.saveFormat](#) property, which is of enum type, and the generated URL will be set to the **src** attribute of the **<source>** tag.

The default **saveFormat** property is set to **Blob** format.

```
`HTML
```

```
<video>
```

```
<source src="blob:http://ej2.syncfusion.com/3ab56a6e-ec0d-490f-85a5-f0aeb0ad8879"
type="video/mp4" >
```

```
</video>
```

```
<video>
```

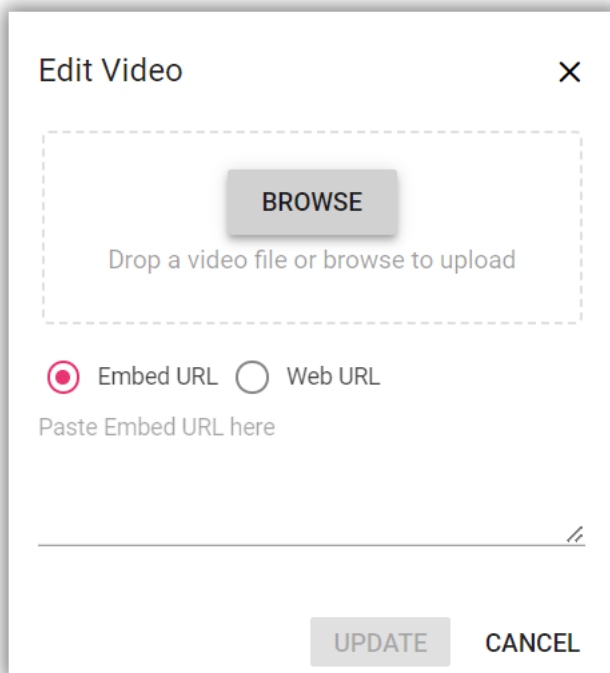
```
<source src="data:video/mp4;base64,iVBORw0KGgoAAAANSUHEUgAAADAAAAAwCAYAAABXAvmHA"
type="video/mp4" >
```

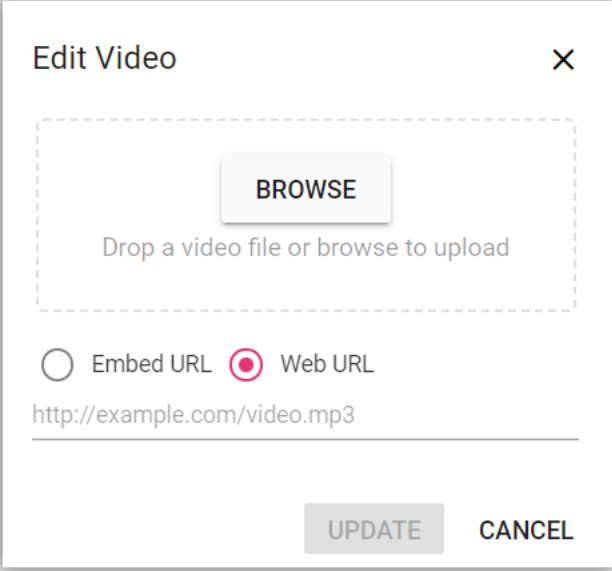
```
</video>
```

```
`
```

Replacing video

Once a video file has been inserted, you can replace it using the Rich Text Editor [quickToolbarSettings](#) `videoReplace` option. You can replace the video file either by using the embedded URL or the web URL and the browse option in the video dialog.



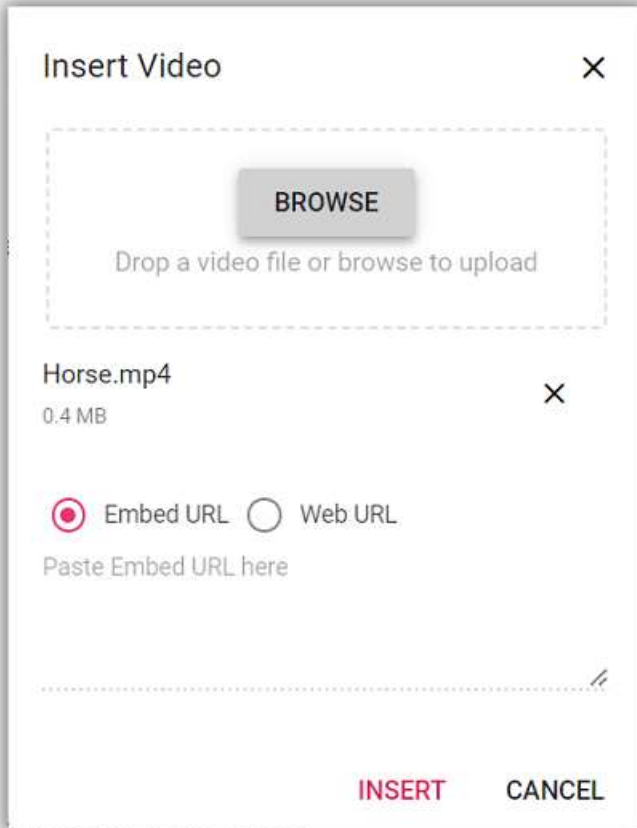


The 'Edit Video' dialog box features a title bar with a close button (X). Inside, there is a dashed rectangular area for video upload. A 'BROWSE' button is centered within this area, with the text 'Drop a video file or browse to upload' below it. Below the dashed area, there are two radio buttons: 'Embed URL' (unselected) and 'Web URL' (selected). A text input field below these buttons contains the URL 'http://example.com/video.mp3'. At the bottom right, there are 'UPDATE' and 'CANCEL' buttons.

Delete video

To remove a video from the Rich Text Editor content, select the video and click the `videoRemove` button from the quick toolbar. It will delete the video from the Rich Text Editor content as well as from the service location if the [insertVideoSettings.removeUrl](#) is given.

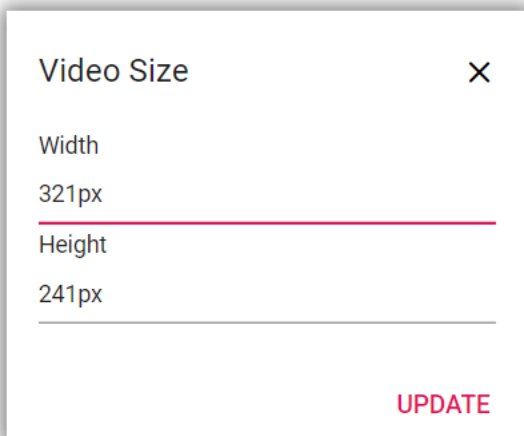
Once you select the video from the local machine, the URL for the video will be generated. You can remove the video from the service location by clicking the cross icon.



Dimension

Set the default width, minWidth, height, and minHeight of the video element, when it is inserted in the Rich Text Editor using the [width](#), [minWidth](#), [height](#), [minHeight](#) properties.

Through the [quickToolbarSettings](#), also you can change the width and height using the **Change Size** button. Once you click on the button, the video size dialog will open as below. In that, specify the width and height of the video in pixels.



Display position

Sets the default display property for the video when it is inserted in the Rich Text Editor using the [insertVideoSettings.layoutOption](#) property. It has two possible options: **Inline** and **Break**. When updating the display positions, it updates the video elements' layout position.

The default **layoutOption** property is set to **Inline**.

```
`typescript
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService } from
'@syncfusion/ej2-angular-richtexteditor';

@Component( {
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertVideoSettings] =
'insertVideoSettings'>
</ejs-richtexteditor>`,
  providers: [
    ToolbarService, QuickToolbarService, LinkService, VideoService, HtmlEditorService,
  ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: [ 'Video' ]
  };
  public insertVideoSettings: Object = {
    layoutOption: 'Break'
  };
}
```

Resize video

The Rich Text Editor has built-in video resizing support, which is enabled for the video elements added. The resize points will appear on each corner of the video when focusing so users can easily resize the video using mouse points or thumb through the resize points. Also, the resize calculation will be done based on the aspect ratio.

You can disable the resize action by configuring **false** for the [insertVideoSettings.resize](#) property.

If the [minWidth](#) and [minHeight](#) properties are configured, the video resizing does not shrink below the specified values.



Rename video before inserting

You can use the [insertVideoSettings](#) property, to specify the server handler to upload the selected video. Then by binding the [fileUploadSuccess](#) event, you can receive the modified file name from the server and update it in the Rich Text Editor's insert video dialog.

`HTML

```
<ejs-richtexteditor id="" [toolbarSettings]='toolbarSettings' [insertVideoSettings] = 'insertVideoSettings'
(fileUploadSuccess) = 'onVideoUploadSuccess($event)' >
```

```
<ng-template #valueTemplate>
```

```
<p>The Rich Text Editor is WYSIWYG ("what you see is what you get") editor useful to create and edit
content, and return the valid <a href="https://ej2.syncfusion.com/home/" target="blank">HTML
markup</a> or <a href="https://ej2.syncfusion.com/home/" target="blank">markdown</a> of the
content</p>
```

```
</ng-template>
```

```
</ejs-richtexteditor>
```

`

`typescript

```
import { Component } from '@angular/core';
import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService, } from
'@syncfusion/ej2-angular-richtexteditor';
@Component({
  selector: 'app-root',
  templateUrl: app.component.html,
  providers: [ ToolbarService, QuickToolbarService, LinkService, VideoService, HtmlEditorService ],
})
export class AppComponent {
```

```

public toolbarSettings: object = {
  items: ['Video'],
};
public insertVideoSettings: Object = {
  saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
  path: "[SERVICEHOSTEDPATH]/Files/"
};
public onVideoUploadSuccess = (args: any) => {
  alert("Get the new file name here");
  if( args.e.currentTarget.getResponseHeader('name') != null ){
    args.file.name = args.e.currentTarget.getResponseHeader('name');
    let fileName : any = document.querySelector(".e-file-name")[0];
    fileName.innerHTML = args.fileData.name.replace(document.querySelectorAll(".e-file-type")[0].innerHTML, "");
    fileName.title = args.fileData.name;
  }
};
}
`

```

To configure server-side handler, refer to the below code.

```

`csharp
int x = 0;
string file;
[AcceptVerbs("Post")]
public void Rename()
{
  try
  {
    var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
    fileName = httpPostedFile.FileName;
    if (httpPostedFile != null)
    {
      var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");

```

```
if (!Directory.Exists(fileSave))
{
    Directory.CreateDirectory(fileSave);
}

var fileName = Path.GetFileName(httpPostedFile.FileName);
var fileSavePath = Path.Combine(fileSave, fileName);
while (System.IO.File.Exists(fileSavePath))
{
    fileName = "rteFiles" + x + "-" + fileName;
    fileSavePath = Path.Combine(fileSave, fileName);
    x++;
}
if (!System.IO.File.Exists(fileSavePath))
{
    httpPostedFile.SaveAs(fileSavePath);
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.Headers.Add("name", fileName);
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusDescription = "File uploaded succesfully";
    Response.End();
}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
```



```

}
}
,

```

Upload video with authentication

You can add additional data with the video uploaded from the Rich Text Editor on the client side, which can even be received on the server side by using the [fileUploading](#) event and its `customFormData` argument, you can pass parameters to the controller action. On the server side, you can fetch the custom headers by accessing the form collection from the current request, which retrieves the values sent using the POST method.

By default, it doesn't support the `UseDefaultCredentials` property, you can manually append the default credentials with the upload request.

```

`typescript
import { Component } from '@angular/core';

import { ToolbarService, QuickToolbarService, LinkService, HtmlEditorService, VideoService, } from
 '@syncfusion/ej2-angular-richtexteditor';

import { UploadingEventArgs } from '@syncfusion/ej2-angular-inputs';

@Component({
  selector: 'app-root',
  template: `<ejs-richtexteditor id=" [toolbarSettings]='toolbarSettings' [insertVideoSettings] =
 'insertVideoSettings' (fileUploading) = 'onVideoUpload($event)' >
</ejs-richtexteditor>`,
  providers: [ ToolbarService, QuickToolbarService, LinkService, VideoService, HtmlEditorService ],
})
export class AppComponent {
  public toolbarSettings: object = {
    items: ['Video'],
  };
  public insertVideoSettings: Object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/SaveFiles",
    path: "[SERVICEHOSTEDPATH]/Files/"
  };
  public onVideoUpload = (args: UploadingEventArgs) => {
    let accessToken = "Authorization_token";
    // adding custom form Data
    args.customFormData = [ { 'Authorization': accessToken }];
  }
}

```

```
};
}
`csharp
public void SaveFiles(IList<IFormFile> UploadFiles)
{
    string currentPath = Request.Form["Authorization"].ToString();
}
`
```

See Also

- [How to edit the quick toolbar settings](#)
- [How to use the link editing option in the toolbar items](#)

Link in Angular Rich text editor component

A hyperlink can be insert into the editor for quick access to the related information. The hyperlink itself can be a text or an image.

Insert link

Point the cursor anywhere within the editor where you would like to insert the link. It is also possible to select a text or an image within the editor and can be converted to the hyperlink. Click the Insert HyperLink tool on the toolbar. The Insert Link Dialog will be open. The dialog has the following options.

`typescript

To use use image and link tool, inject `ImageService`, `LinkService` in the provider section of `AppModule`.

`

Options	Description
Web Address	Type or paste the destination for the link you are creating
Display Text	Type or edit the required text that you want to display text for the link
Tooltip	To display additional helpful information when you place the pointer on the hyperlink, type the required text in the "Tooltip" field.
Open Link in New Window	Specify whether, the given link will be open in new window or not

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
```

```
enableRipple(true);
/**
 * Rich Text Editor Link sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from
 '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]
='tools'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public tools = {
    items: ['CreateLink', 'RemoveLink']
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The Rich Text Editor link tool validates the URLs, as you type in the Web Address. URLs considered invalid will be highlighted with red color by clicking the insert button in the **Insert Link** dialog.

Remove Link

If you want to remove a hyperlink from a text or image, select the text or image with the hyperlink and click “Remove Hyperlink” tool from toolbar. It will keep the text or image.

Auto-link

When you type URL and Enter key to the Rich Text Editor, the typed URL will be automatically changed into the hyperlink.

Manipulation

Add the custom tools on the selected link inside the Rich Text Editor through the quick toolbar.



The quick toolbar for the Link has the following options.

Tools	Description
-----	-----

- | Open | The given link page, will be open in new window. |
- | Edit Link | Used to edit the link in the Rich Text Editor content. |
- | Remove Link | Used to remove link from the content of Rich Text Editor. |
- | Custom Tool | Used to add the custom options in the quick toolbar. |

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor Link sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public tools = {
    items: ['CreateLink', 'RemoveLink']
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Table in Angular Rich text editor component

RichTextEditor allows to insert table of content in edit panel and provides an options to add, edit and remove the table as well as perform other table related action. For inserting the table to the Rich Text Editor, the following list of options have been provided in the [tableSettings](#)

- | Options | Description | Default Value |
|---------|-------------|---------------|
| ----- | ----- | ----- |

- | [minWidth](#) | Sets the default minWidth of the table. | 0 |
- | [maxWidth](#) | Sets the default maxWidth of the table. | null |
- | [resize](#) | Enable resize feature in table. | true |
- | [styles](#) | This is an array of key value pair, on each pair, key should be name of styling and value is class name. this list will be shown on quick toolbar options to change the styles of table on designing like dashed, double bordered. | [TableStyleItems](#) |
- | [width](#) | Sets the default width of the table. | 100% |

To use Table feature, inject **TableService** in the provider section of **AppModule**.

Insert table

Using the **table** toolbar option, select a number of rows and columns to be inserted over the table grid and insert table into Rich Text Editor content using the mouse.

Tables can also be inserted through the **Insert Table** option in the pop-up where the number of rows and columns can be provided manually, and this is the default way in devices.

In the following sample, the table has been injected from table module.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
TableService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='iframeRTE' [toolbarSettings]='tools'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor component is WYSIWYG ("what you see
is what you get") editor
      that provides the best user experience to create and update
the content.
      Users can format their content using standard toolbar
commands.</p>
      <p><b>Key features:</b></p>
      <ul><li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;;
modes</p></li>
      <li><p>Capable of handling markdown editing.</p></li>
      <li><p>Contains a modular library to load the necessary
functionality on demand.</p></li>
      <li><p>Provides a fully customizable toolbar.</p></li>
      <li><p>Provides HTML view to edit the source directly for
developers.</p></li>
```

```

        <li><p>Supports third-party library integration.</p></li>
        <li><p>Allows preview of modified content before saving
it.</p></li>
        <li><p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p></li>
        <li><p>Contains undo/redo manager.</p></li>
        <li><p>Creates bulleted and numbered lists.</p></li>
        </ul>
    </ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
TableService]
})
export class AppComponent {
    public tools: object = {
        items: ['CreateTable']
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Quick Toolbar

Quick toolbar will open by clicking on the table. It has different sets of commands to be performed on the table which increases the feasibility to edit the table easily.

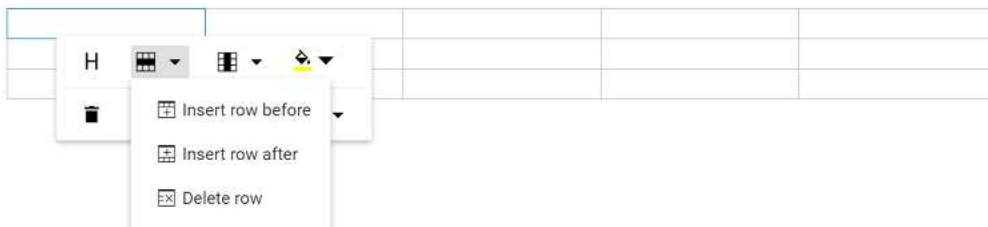
To use quick toolbar feature, inject **QuickToolbarService** in the provider section of **AppModule**.

Table Header

Table Header command is available with quick toolbar option through which the header row can be added or removed from the inserted table. The following image illustrates the table header.

Insert Rows

Rows can be inserted above or below the required table cell through the quick toolbar. Also, focused row can be deleted. The following screenshot shows the available options of the row item.



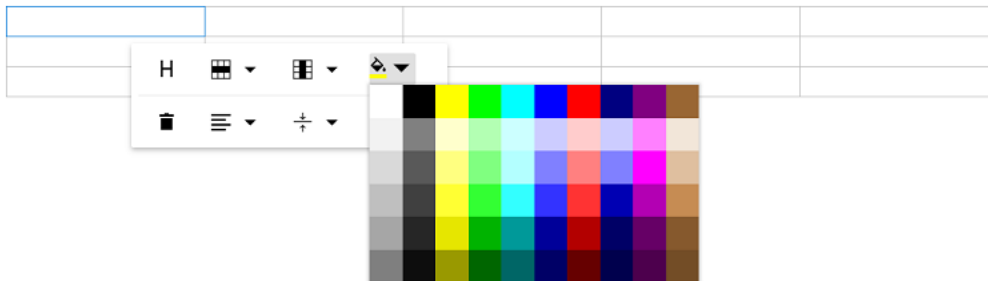
Insert Columns

Columns can be inserted to the left or right side of the required table cell through the quick toolbar. Also, the focused column can be deleted. The following screenshot shows the available options in inserting column item.



Set Color

The background color can be set for each table cell through the **background color** command available in quick toolbar.



Delete Table

Using the delete item in the quick toolbar, users can delete the entire table.

Vertical Align

Text inside the table can be aligned to top, middle, or bottom using the **tableCellVerticalAlign** tool of the quick toolbar.



Horizontal Align

Text inside the table can be aligned left, right, or center using the **tableCellHorizontalAlign** tool of the quick toolbar.

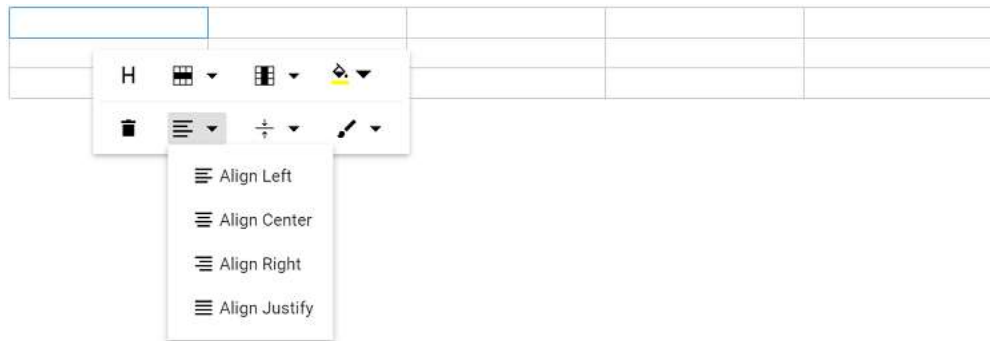


Table Styles

Table styles provided for class name should be appended to a table element. It helps to design the table in specific CSS styles when inserting in the editor.

By Default, provides Dashed border and Alternate rows.

Dashed border: Applies the dashed border to the table.

Alternate border: Applies the alternative background to the table.

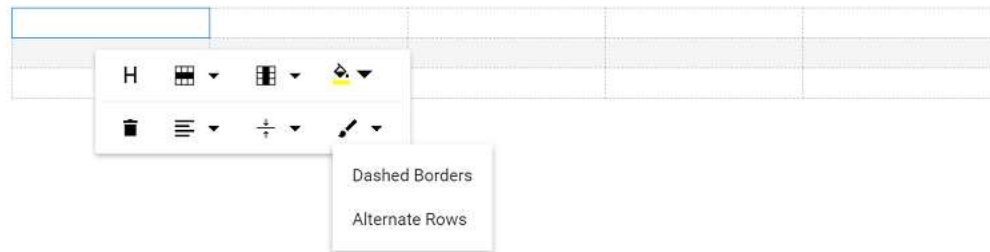


Table Properties

Sets the default width of the table when it is inserted in the Rich Text Editor using the width of `tableSettings`.

Using the quick toolbar, users can change the width, cell padding, and cell spacing in the selected table using the properties option.

Edit Table

Width

1,189

Cell Padding

0

Cell Spacing

0

UPDATE

CANCEL

Table cell merge and split

The Rich Text Editor allows users to change the appearance of the tables by splitting or merging the table cells.

TableCell item should be configured in the Table [quickToolbarSettings](#) Property to show the merge/split icons while selecting the table cells

Table cell merge

The table cell merge feature allows you to merge two or more row and column cells into a single cell with its contents.

The following image explains the table merge action.

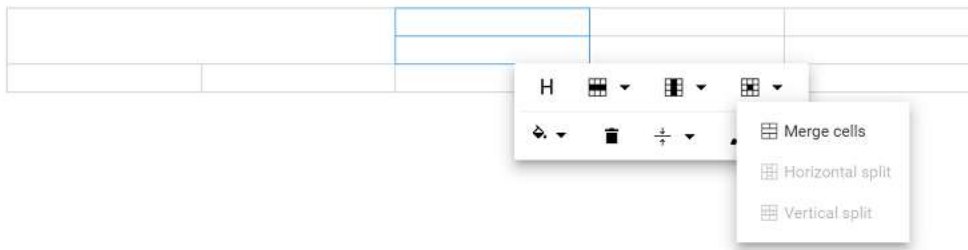


Table cell split

The table cell split feature allows you to a selected cell can be split both horizontally and vertically.

The following image explains the table split action.



APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
    import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
TableService, QuickToolbarSettingsModel } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
imports: [

        RichTextEditorAllModule,
        DialogModule
    ],
standalone: true,
selector: 'app-root',
template: `<ejs-richtexteditor id='iframeRTE' [toolbarSettings]='tools'
[quickToolbarSettings]='quickToolbarSettings'>
    <ng-template #valueTemplate>
        <p>The Rich Text Editor component is WYSIWYG ("what you see
is what you get") editor
        that provides the best user experience to create and update
the content.
        Users can format their content using standard toolbar
commands.</p>
        <p><b>Key features:</b></p>
        <ul><li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;;
modes</p></li>
        <li><p>Capable of handling markdown editing.</p></li>
        <li><p>Contains a modular library to load the necessary
functionality on demand.</p></li>
        <li><p>Provides a fully customizable toolbar.</p></li>
        <li><p>Provides HTML view to edit the source directly for
developers.</p></li>
        <li><p>Supports third-party library integration.</p></li>
        <li><p>Allows preview of modified content before saving
it.</p></li>
        <li><p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p></li>
        <li><p>Contains undo/redo manager.</p></li>
        <li><p>Creates bulleted and numbered lists.</p></li>
        </ul>
    </ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
TableService]
})
export class AppComponent {
    public tools: object = {
        items: ['CreateTable'],
    };
    public quickToolbarSettings: QuickToolbarSettingsModel = {
        table: ['TableHeader', 'TableRows', 'TableColumns', 'TableCell', '-',
'BackgroundColor', 'TableRemove', 'TableCellVerticalAlign', 'Styles']

```

```
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Emoji Picker in Angular RichTextEditor component

An emoji picker is a tool that allows users to add emojis or emoticons to their text easily. Typically, it is a small window or panel that displays a variety of emojis arranged in different categories, such as smileys, animals, food, and so on. Users can select the desired emoji by clicking on it or by typing its name in a search bar.

Enabling the toolbar option and custom emojis.

Add the **EmojiPicker** tool to the toolbar of the RichTextEditor by utilizing the **toolbarSettings** [items](#) property.

By default, a predefined set of emojis is configured. However, you can customize these icons according to your needs. To achieve this, utilize the [emojiPickerSettings](#) property.

```
`ts
```

```
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { RichTextEditorComponent, ToolbarService, HtmlEditorService, ImageService,
QuickToolbarService, LinkService, EmojiPickerService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  selector: 'control-content',
  templateUrl: 'insert-emoticons.html',
  encapsulation: ViewEncapsulation.None,
  styleUrls: ['style.css'],
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService, QuickToolbarService,
EmojiPickerService]
})
export class AppComponent {
  @ViewChild('emojiPickerRTE')
  public emojiPickerRTE: RichTextEditorComponent;
  public toolbarSettings: ToolbarSettingsModel = {
    items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments', 'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode', 'EmojiPicker', '|', 'Undo', 'Redo'
]
}
```

```

};
public emojiPickerSettings: EmojiSettingsModel =
{
  iconsSet: [{name: 'Smilies & People', code: '1F600', iconCss: 'e-emoji',
  icons: [{ code: '1F600', desc: 'Grinning face' },
  { code: '1F603', desc: 'Grinning face with big eyes' },
  { code: '1F604', desc: 'Grinning face with smiling eyes' },
  { code: '1F606', desc: 'Grinning squinting face' },
  { code: '1F605', desc: 'Grinning face with sweat' },
  { code: '1F602', desc: 'Face with tears of joy' },
  { code: '1F923', desc: 'Rolling on the floor laughing' },
  { code: '1F60A', desc: 'Smiling face with smiling eyes' }]
}, {
  name: 'Animals & Nature', code: '1F435', iconCss: 'e-animals',
  icons: [{ code: '1F436', desc: 'Dog face' },
  { code: '1F431', desc: 'Cat face' },
  { code: '1F42D', desc: 'Mouse face' },
  { code: '1F439', desc: 'Hamster face' },
  { code: '1F430', desc: 'Rabbit face' },
  { code: '1F98A', desc: 'Fox face' }]
}, {
  name: 'Food & Drink', code: '1F347', iconCss: 'e-food-and-drinks',
  icons: [{ code: '1F34E', desc: 'Red apple' },
  { code: '1F34C', desc: 'Banana' },
  { code: '1F347', desc: 'Grapes' },
  { code: '1F353', desc: 'Strawberry' },
  { code: '1F35E', desc: 'Bread' },
  { code: '1F950', desc: 'Croissant' },
  { code: '1F955', desc: 'Carrot' },
  { code: '1F354', desc: 'Hamburger' }]
}, {
  name: 'Activities', code: '1F383', iconCss: 'e-activities',
  icons: [{ code: '26BD', desc: 'Soccer ball' },

```

```

{ code: '1F3C0', desc: 'Basketball' },
{ code: '1F3C8', desc: 'American football' },
{ code: '26BE', desc: 'Baseball' },
{ code: '1F3BE', desc: 'Tennis' },
{ code: '1F3D0', desc: 'Volleyball' },
{ code: '1F3C9', desc: 'Rugby football' }}
}, {
name: 'Travel & Places', code: '1F30D', iconCss: 'e-travel-and-places',
icons: [{ code: '2708', desc: 'Airplane' },
{ code: '1F697', desc: 'Automobile' },
{ code: '1F695', desc: 'Taxi' },
{ code: '1F6B2', desc: 'Bicycle' },
{ code: '1F68C', desc: 'Bus' }]
}, {
name: 'Objects', code: '1F507', iconCss: 'e-objects', icons: [{ code: '1F4A1', desc: 'Light bulb' },
{ code: '1F526', desc: 'Flashlight' },
{ code: '1F4BB', desc: 'Laptop computer' },
{ code: '1F5A5', desc: 'Desktop computer' },
{ code: '1F5A8', desc: 'Printer' },
{ code: '1F4F7', desc: 'Camera' },
{ code: '1F4F8', desc: 'Camera with flash' },
{ code: '1F4FD', desc: 'Film projector' }]
}, {
name: 'Symbols', code: '1F3E7', iconCss: 'e-symbols', icons: [{ code: '274C', desc: 'Cross mark' },
{ code: '2714', desc: 'Check mark' },
{ code: '26A0', desc: 'Warning sign' },
{ code: '1F6AB', desc: 'Prohibited' },
{ code: '2139', desc: 'Information' },
{ code: '267B', desc: 'Recycling symbol' },
{ code: '1F6AD', desc: 'No smoking' }]
}}
}
}

```

Additionally, you have the option to customize the icons of toolbar items using the [iconCss](#) and [code](#) properties. The `iconCSS` property allows you to define a custom CSS class for the toolbar item icon, while the `code` property enables you to specify the Unicode character code for the icon.

When both `iconCSS` and `code` properties are provided, the `iconCSS` property takes precedence in determining the appearance of the toolbar item icon.

Additionally, you have the option to enhance the user experience by implementing a filtering feature for efficiently managing a large dataset of emojis. By setting the [showSearchBox](#) property to true (which is the default value), users will be able to utilize a search box to filter the displayed emojis according to their preferences.

The following code example shows how to add the emoji picker tool in the RichTextEditor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
/**
 * Rich Text Editor Custom-Toolbar Sample
 */
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { RichTextEditorComponent, ToolbarService, HtmlEditorService,
ImageService, QuickToolbarService, LinkService, EmojiPickerService } from
 '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor [toolbarSettings]='toolbarSettings'>
<ng-template #valueTemplate>
  <p>An emoji picker in a Rich Text Editor is a tool that allows users to
easily add emojis or emoticons to their text.</p>
  <p>Typically, it is a small window or panel that displays a variety of
emojis, arranged in different categories, such as smileys, animals, food, and
so on. Users can select the desired emoji by clicking on it or by typing its
name in a search bar.</p>
</ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService, EmojiPickerService]
} )
export class AppComponent {
  public toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode',
'EmojiPicker', '|', 'Undo', 'Redo']
```

```

    }
  };
}

```

MAIN.TS

```

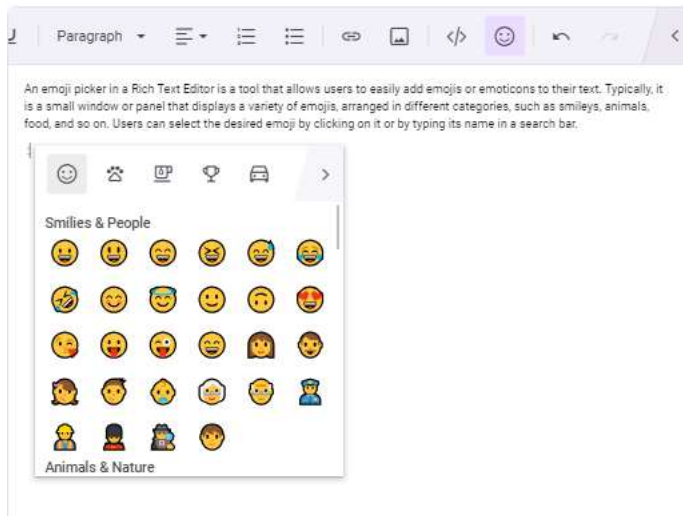
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Rich Text Editor features are segregated into individual feature-wise modules. In this demo, we have used the following injectable service `EmojiPickerService` in the `@NgModule.providers` section.

Using the shortcut key to open the emoji picker

Quickly access the emoji picker by pressing the colon (:) key while typing a word prefix in an editor, allowing instant emoji selection and display. Moreover, continue typing in the editor after the colon (:) to filter and refine your search for the desired emojis.



Navigating and selecting emojis using the keyboard

The emoji picker popup offers keyboard navigation options, allowing you to move the emoji focus from one emoji to another. The following keys are used for navigation:

Arrow keys: Use the arrow keys (up, down, left, right) to move the emoji focus in the corresponding direction.

Enter: Press Enter key to select the currently focused emoji.

Escape: Press Escape to close the emoji picker popup without selecting an emoji.

Markdown in Angular Rich text editor component

When you format the word in Markdown format, you should add Markdown syntax to the word to indicate the words and phrases that looks different from each other.

Rich Text Editor supports markdown editing when the `editorMode` set as **markdown** and using both *keyboard interaction* and *toolbar action*, you can apply the formatting to text.

To use quick Markdown editing feature, inject `MarkdownEditorService` in the provider section of [Link to the Video](#).

To build a markdown editor using the Angular Rich Text Editor, refer to the video below.

Supported Commands

The Angular Markdown editor supports the following commands to format the markdown content:

Commands	Syntax	Description
	----- ----- -----	
Bold	Sample content for bold text .	For bold, add <code>or</code> to front and back of the text. For order list, precede each line with a number.
Italic	Sample content for <i>Italic text</i> .	For Italic, add <code>* or _</code> to front and back of the text.
Bold and Italics	Sample content for <i>bold and Italic text</i> .	For bold and Italics, add <code>*</code> to the front and back of the text.
Heading 1	<h1>Heading 1 content</h1>	For heading 1, add <code>#</code> to start of the line.
Heading 2	<h2>Heading 2 content</h2>	For heading 2, add <code>##</code> to start of the line.
Heading 3	<h3>Heading 3 content</h3>	For heading 3, add <code>###</code> to start of the line.
Heading 4	<h4>Heading 4 content</h4>	For heading 4, add <code>####</code> to start of the line.
Heading 5	<h5>Heading 5 content</h5>	For heading 5, add <code>#####</code> to start of the line.
Heading 6	<h6>Heading 6 content</h6>	For heading 6, add <code>#####</code> to start of the line.
Line Break	First line Second line	For line break, press enter two times (or) add <code>
</code> in between the first and the second line.
Blockquotes	> Blockquotes text	For blockquotes, add <code>></code> to start of the line.
Strike Through	Sample content for strike through text .	For strike through, add <code>~~</code> to front and back of the text.
Code (Single line)	<code>Single line code</code>	For single line code, add <code>`</code> to front and back of the text.
Code block (Multi Line)	<pre>\\
Multi line code text
Multi line code text
\\</pre>	For multiple line code, add <code>\\`</code> in the new line before and after the content.
Subscript	_{Subscript text}	For subscript, add <code><sub></code> to the front and <code></sub></code> to the back of the text.
Superscript	^{Superscript text}	For superscript, add <code><sup></code> to the front and <code></sup></code> to the back of the text.
Ordered List	1. First 1. Second	For ordered list, preceding one or more lines of text with <code>1.</code>
Unordered List	First second	For unordered list, preceding one or more lines of text with <code>*</code> .
Links	Link text without title text Link text Link text with title text [Link text](URL , "title text")	Create an inline link by wrapping link text in brackets [], and then

wrapping the URL as first parameter and title as second parameter in the parentheses ().
Note: The title text is optional, if needed it can be given manually. |

| Table | | Heading 1 | Heading 2 |
|-----|-----|
| Col A1 | Col A2 |
| Col B1 | Col B2 | | Create a table using the pipes and underscores as given in the syntax to create 2 x 2 table. |

| Horizontal Line | *(three asterix in new line)
(or)
(three underscores in new line)* |
For horizontal line, add or to the start of the new line. |

| Image | | Create an image by wrapping the image source in parentheses (). |

| Image with alternate text | ![alternate text](URL path) | Create an image with alternate text by wrapping an alternative text in brackets [], and then link of the image source in parentheses ().
Note: When inserting the image using toolbar, the alternate text cannot be provided that needs to be given manually. |

| Escape tick marks supported | Sample text content with **bold and** not bold **text can be in the same line.** | In the syntax, the whole content is made as bold where the content not bold can be made as normal text by adding the bold syntax to the start and end of the respective text. Likewise you can do the same for various inline commands. |

| Escape Character | \ (any syntax) | Escape any markdown syntax by prefix \ to the syntax.
 Example:
Bold text |

| HTML Entities | Copyright: © - ©
Trade mark: ™ - ™
Registered: ® - ®
Ampersand: & - &
Less than: < - <
Greater than: > - > | For HTML entities, add & and ; to the front and back of the respective entities. |

The above listed commands alone are supported in Syncfusion Markdown editor. For other unsupported commands, you can achieve using the HTML tags in Markdown editor. The foot notes, definitions, math, and check list markdown syntax are also not supported.

Markdown to HTML

The Rich Text Editor allows you to preview markdown changes immediately using preview. In this sample, the third-party library [Marked](#) is used to convert markdown into HTML content.

This sample demonstrates how to preview markdown changes in Rich Text Editor. Type or edit the display text, and apply format to view the preview of markdown. The [actionComplete](#) event can be used to convert Markdown to HTML.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor Markdown Preview demo
 */
```

```

import { Component, ViewChild } from '@angular/core';
import { addClass, removeClass, Browser } from '@syncfusion/ej2-base';
import { RichTextEditorComponent, ToolbarService, LinkService } from '@syncfusion/ej2-angular-richtexteditor';
import { ImageService, MarkdownEditorService } from '@syncfusion/ej2-angular-richtexteditor';
import { createElement, KeyboardEventArgs, isNullOrUndefined } from '@syncfusion/ej2-base';
import * as Marked from 'marked';

@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='mdPreview' #mdPreview
[toolbarSettings]='tools' [editorMode]='mode' (created)='onCreate()'
(actionComplete)="actionComplete($event)">
    <ng-template #valueTemplate>
      In Rich Text Editor , you click the toolbar buttons to format the
      words and the changes are visible immediately.
      Markdown is not like that. When you format the word in Markdown
      format, you need to
      add Markdown syntax to the word to indicate which words
      and phrases should look different from each other.
      Rich Text Editor supports markdown editing when the editorMode
set as **markdown** and using both *keyboard interaction* and *toolbar
action*, you can apply the formatting to text.
      You can add our own custom formation syntax for the Markdown
      formation, [sample link](https://ej2.syncfusion.com/home/).
      The third-party library <b>Marked</b> is used in this sample to
      convert markdown into HTML content.
    </ng-template>
  </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService,
  MarkdownEditorService]
}))
export class AppComponent {
  @ViewChild('mdPreview')
  public rteObj?: RichTextEditorComponent;
  public textArea?: HTMLTextAreaElement;
  public mdsources?: HTMLCollection;
  public mdSplit?: HTMLCollection;
  public htmlPreview?: HTMLCollection;
  public tools: object = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
    'UnorderedList', '|', 'CreateLink', 'Image', '|',
    {
      tooltipText: 'Preview',
      template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-icons"></span></button>'
    }, {
      tooltipText: 'Split Editor',
      template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +

```

```

        '<span class="e-btn-icon e-view-side e-icons"></span></button>'
        }, 'FullScreen', '|', 'Undo', 'Redo']
    };
    public mode: string = 'Markdown';
    public onCreate(): void {
        let script = document.createElement('script');
        script.src = 'https://cdn.jsdelivr.net/npm/marked/marked.min.js';
        document.head.appendChild(script);
        this.textArea = (this.rteObj!.contentModule as any).getEditPanel() as
        HTMLTextAreaElement;
        this.textArea.addEventListener('keyup', () => {
            this.markDownConversion();
        });
        this.mdsource = document.getElementById('preview-code') as any;
        this.mdsource!.addEventListener('click', (e: MouseEvent) => {
            this.fullPreview({ mode: true, type: 'preview' });
            if ((e.target as HTMLElement).parentElement!.classList.contains('e-
            active')) {
                this.rteObj!.disableToolbarItem(['Bold', 'Italic',
                'StrikeThrough', 'OrderedList',
                'UnorderedList', 'CreateLink', 'Image']);
                (e.target as
                HTMLElement)!.parentElement!.parentElement!.nextElementSibling!.classList.add
                ('e-overlay');
            } else {
                this.rteObj!.enableToolbarItem(['Bold', 'Italic',
                'StrikeThrough', 'OrderedList',
                'UnorderedList', 'CreateLink', 'Image']);
                (e.target as
                HTMLElement)!.parentElement!.parentElement!.nextElementSibling!.classList.remo
                ve('e-overlay');
            }
        });
        this.mdSplit = document.getElementById('MD_Preview') as any;
        this.mdSplit!.addEventListener('click', (e: MouseEvent) => {
            if (this.rteObj!.element.classList.contains('e-rte-full-screen')) {
                this.fullPreview({ mode: true, type: '' });
            }
            this.mdsource!.classList.remove('e-active');
            this.rteObj!.showFullScreen();
        });
    }
    public actionComplete(e: any): void {
        if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
            this.fullPreview({ mode: true, type: '' });
        } else if (!this.mdSplit!.parentElement!.classList.contains('e-overlay'))
        {
            if (e.targetItem === 'Minimize') {
                this.textArea!.style.display = 'block';
                this.textArea!.style.width = '100%';
                if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
            }

            this.mdSplit!.classList.remove('e-active');
            this.mdsource!.classList.remove('e-active');
        }
        this.markDownConversion();
    }
}

```

```

}
public markDownConversion(): void {
  if (this.mdSplit!.classList.contains('e-active')) {
    let id: string = this.rteObj!.getID() + 'html-preview';
    let htmlPreview: HTMLElement = this.rteObj!.element.querySelector('#' + id) as HTMLElement;
    debugger
    htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as any).getEditPanel() as HTMLTextAreaElement).value);
  }
}
public fullPreview(e: { [key: string]: string | boolean }): void {
  let id: string = this.rteObj!.getID() + 'html-preview';
  this.htmlPreview = this.rteObj!.element.querySelector('#' + id) as HTMLElement;
  if ((this.mdsouce!.classList.contains('e-active') || this.mdSplit!.classList.contains('e-active')) && e['mode']) {
    this.mdsouce!.classList.remove('e-active');
    this.mdSplit!.classList.remove('e-active');
    this.textArea!.style.display = 'block';
    this.textArea!.style.width = '100%';
    this.htmlPreview.style.display = 'none';
  } else {
    this.mdsouce!.classList.add('e-active');
    this.mdSplit!.classList.add('e-active');
    if (!this.htmlPreview) {
      this.htmlPreview = createElement('div', { className: 'e-content'
    });
    this.htmlPreview.id = id;
    this.textArea!.parentNode!.appendChild(this.htmlPreview);
  }
  if (e['type'] === 'preview') {
    this.textArea!.style.display = 'none';
    this.htmlPreview.classList.add('e-pre-source');
  } else {
    this.htmlPreview.classList.remove('e-pre-source');
    this.textArea!.style.width = '50%';
  }
  this.htmlPreview.style.display = 'block';
  this.htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as any).getEditPanel() as HTMLTextAreaElement).value);
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Table

Rich Text Editor allows to insert Markdown table in edit panel with 2 X 2 rows and columns along with the heading.

To use table tool, add the **CreateTable** item in toolbar items.

Insert table

To insert the table in Rich Text Editor's content area, click the insert table icon in the toolbar option.

Please refer the below sample and code snippets to add the table in Markdown editor

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor Markdown demo with table
 */
import { Component, ViewChild } from '@angular/core';
import { RichTextEditorComponent, ToolbarService, LinkService } from '@syncfusion/ej2-angular-richtexteditor';
import { ImageService, MarkdownEditorService } from '@syncfusion/ej2-angular-richtexteditor';
import { createElement, KeyboardEventArgs, isNullOrUndefined } from '@syncfusion/ej2-base';
import * as Marked from 'marked';
@Component({
  imports: [
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='mdPreview' #mdPreview
[toolbarSettings]='tools' [editorMode]='mode' (created)='onCreate()'
(actionComplete)="actionComplete($event)">
<ng-template #valueTemplate>
  In Rich Text Editor , you click the toolbar buttons to format
the words and the changes are visible immediately.
  Markdown is not like that. When you format the word in
Markdown format, you need to
  add Markdown syntax to the word to indicate which words
and phrases should look different from each other.
  Rich Text Editor supports markdown editing when the
editorMode set as **markdown** and using
both *keyboard interaction* and *toolbar action*, you can
apply the formatting to text.
  You can add our own custom formation syntax for the Markdown
formation, [sample link](https://ej2.syncfusion.com/home/).
  The third-party library <b>Marked</b> is used in this sample
to convert markdown into HTML content.
</ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, MarkdownEditorService]
})
export class AppComponent {
  @ViewChild('mdPreview')
```

```

public rteObj?: RichTextEditorComponent;
public textArea?: HTMLTextAreaElement;
public mdsource?: HTMLElement;
public mdSplit?: HTMLElement;
public htmlPreview?: HTMLElement;
public tools: object = {
  items: ['Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
    'UnorderedList', '|', 'CreateLink', 'Image', 'CreateTable', '|',
    {
      tooltipText: 'Preview',
      template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
        '<span class="e-btn-icon e-md-preview e-icons"></span></button>'
    }, {
      tooltipText: 'Split Editor',
      template: '<button id="MD_Preview" class="e-tbar-btn e-control e-
btn e-icon-btn">' +
        '<span class="e-btn-icon e-view-side e-icons"></span></button>'
    }, 'FullScreen', '|', 'Undo', 'Redo']
};
public mode: string = 'Markdown';
public onCreate(): void {
  let script = document.createElement('script');
  script.src = 'https://cdn.jsdelivr.net/npm/marked/marked.min.js';
  document.head.appendChild(script);
  this.textArea = (this.rteObj!.contentModule as any).getEditPanel() as
HTMLTextAreaElement;
  this.textArea.addEventListener('keyup', () => {
    this.markDownConversion();
  });
  this.mdsource = document.getElementById('preview-code') as any;
  this.mdsource!.addEventListener('click', (e: MouseEvent) => {
    this.fullPreview({ mode: true, type: 'preview' });
    if ((e.target as HTMLElement).parentElement!.classList.contains('e-
active')) {
      this.rteObj!.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
'UnorderedList', 'CreateLink', 'Image', 'CreateTable']);
      (e.target as
HTMLElement).parentElement!.parentElement!.nextElementSibling!.classList.add(
'e-overlay');
    } else {
      this.rteObj!.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
'UnorderedList', 'CreateLink', 'Image', 'CreateTable']);
      (e.target as
HTMLElement).parentElement!.parentElement!.nextElementSibling!.classList.remo
ve('e-overlay');
    }
  });
  this.mdSplit = document.getElementById('MD_Preview') as any;
  this.mdSplit!.addEventListener('click', (e: MouseEvent) => {
    if (this.rteObj!.element.classList.contains('e-rte-full-screen')) {
      this.fullPreview({ mode: true, type: '' });
    }
    this.mdsource!.classList.remove('e-active');
    this.rteObj!.showFullScreen();
  });
}

```

```

    });
  }
  public actionComplete(e: any): void {
    if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
      this.fullPreview({ mode: true, type: '' });
    } else if (!this.mdSplit!.parentElement!.classList.contains('e-overlay'))
    {
      if (e.targetItem === 'Minimize') {
        this.textArea!.style.display = 'block';
        this.textArea!.style.width = '100%';
        if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
      }

      this.mdSplit!.classList.remove('e-active');
      this.mdsources!.classList.remove('e-active');
    }
    this.markDownConversion();
  }
}
public markDownConversion(): void {
  if (this.mdSplit!.classList.contains('e-active')) {
    let id: string = this.rteObj!.getID() + 'html-preview';
    let htmlPreview: HTMLElement = this.rteObj!.element.querySelector('#' +
+ id) as HTMLElement;
    htmlPreview.innerHTML = marked.parse(((this.rteObj!.contentModule as
any).getEditPanel() as HTMLTextAreaElement).value);
  }
}
public fullPreview(e: { [key: string]: string | boolean }): void {
  let id: string = this.rteObj!.getID() + 'html-preview';
  this.htmlPreview = this.rteObj!.element.querySelector('#' + id) as
HTMLElement;
  if ((this.mdsources!.classList.contains('e-active') ||
this.mdSplit!.classList.contains('e-active')) && e['mode']) {
    this.mdsources!.classList.remove('e-active');
    this.mdSplit!.classList.remove('e-active');
    this.textArea!.style.display = 'block';
    this.textArea!.style.width = '100%';
    this.htmlPreview.style.display = 'none';
  } else {
    this.mdsources!.classList.add('e-active');
    this.mdSplit!.classList.add('e-active');
    if (!this.htmlPreview) {
      this.htmlPreview = createElement('div', { className: 'e-content'
    });
    }
    this.htmlPreview.id = id;
    this.textArea!.parentNode!.appendChild(this.htmlPreview);
  }
  if (e['type'] === 'preview') {
    this.textArea!.style.display = 'none';
    this.htmlPreview.classList.add('e-pre-source');
  } else {
    this.htmlPreview.classList.remove('e-pre-source');
    this.textArea!.style.width = '50%';
  }
  this.htmlPreview.style.display = 'block';

```

```

        this.htmlPreview.innerHTML =
marked.parse(((this.rteObj!.contentModule as any).getEditPanel() as
HTMLTextAreaElement).value);
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Changing table constants

The Markdown table constants can be changed for the table heading and the column names.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor changing table constants (markdown)
 */
import { Component, ViewChild } from '@angular/core';
import { RichTextEditorComponent, ToolbarService, LinkService } from
 '@syncfusion/ej2-angular-richtexteditor';
import { ImageService, MarkdownEditorService } from '@syncfusion/ej2-angular-richtexteditor';
import { createElement, KeyboardEventArgs, isNullOrUndefined } from
 '@syncfusion/ej2-base';
import { L10n } from '@syncfusion/ej2-base';
import * as Marked from 'marked';
L10n.load({
  'en-US': {
    'richtexteditor': {
      'TableHeadingText': 'Header',
      'TableColText': 'Cell'
    }
  }
});
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='mdPreview' #mdPreview
[toolbarSettings]='tools' [editorMode]='mode'
(created)='onCreate()' (actionComplete)="actionComplete($event)">
<ng-template #valueTemplate>

```


In Rich Text Editor , you click the toolbar buttons to format the words and the changes are visible immediately.

Markdown **is** not like that. When you format the word **in** Markdown format, you need to

add Markdown syntax to the word to indicate which words and phrases should look different **from** each other.

Rich Text Editor supports markdown editing when the editorMode **set as** ****markdown**** and **using**

both ***keyboard interaction*** and ***toolbar action***, you can apply the formatting to text.

You can **add** our own custom formation syntax **for** the Markdown formation, [sample link](<https://ej2.syncfusion.com/home/>).

The third-party library `Marked` **is** used **in this** sample to convert markdown **into** HTML content.

```

</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, MarkdownEditorService]
})
export class AppComponent {
  @ViewChild('mdPreview')
  public rteObj?: RichTextEditorComponent;
  public textArea?: HTMLTextAreaElement;
  public mdsource?: HTMLInputElement;
  public mdSplit?: HTMLInputElement;
  public htmlPreview?: HTMLInputElement;
  public tools: object = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|', 'Formats', 'OrderedList',
      'UnorderedList',
      '|', 'CreateLink', 'Image', 'CreateTable', '|',
      {
        tooltipText: 'Preview',
        template: '<button id="preview-code" class="e-tbar-btn e-control e-btn e-icon-btn">' +
          '<span class="e-btn-icon e-md-preview e-icons"></span></button>'
      }, {
        tooltipText: 'Split Editor',
        template: '<button id="MD_Preview" class="e-tbar-btn e-control e-btn e-icon-btn">' +
          '<span class="e-btn-icon e-view-side e-icons"></span></button>'
      }, 'FullScreen', '|', 'Undo', 'Redo']
  };
  public mode: string = 'Markdown';
  public onCreate(): void {
    let script = document.createElement('script');
    script.src = 'https://cdn.jsdelivr.net/npm/marked/marked.min.js';
    document.head.appendChild(script);
    this.textArea = (this.rteObj!.contentModule as any).getEditPanel() as HTMLTextAreaElement;
    this.textArea.addEventListener('keyup', () => {
      this.markDownConversion();
    });
    this.mdsource = document.getElementById('preview-code') as any;
    this.mdsource!.addEventListener('click', (e: MouseEvent) => {
      this.fullPreview({ mode: true, type: 'preview' });
      if ((e.target as HTMLInputElement).parentElement!.classList.contains('e-active')) {

```

```

        this.rteObj!.disableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
        'UnorderedList', 'CreateLink', 'Image', 'CreateTable']);
        (e.target as
HTMLElement).parentElement!.parentElement!.nextElementSibling!.classList.add(
'e-overlay');
    } else {
        this.rteObj!.enableToolbarItem(['Bold', 'Italic',
'StrikeThrough', 'OrderedList',
        'UnorderedList', 'CreateLink', 'Image', 'CreateTable']);
        (e.target as
HTMLElement).parentElement!.parentElement!.nextElementSibling!.classList.remo
ve('e-overlay');
    }
});
this.mdSplit = document.getElementById('MD_Preview') as any;
this.mdSplit!.addEventListener('click', (e: MouseEvent) => {
    if (this.rteObj!.element.classList.contains('e-rte-full-screen')) {
        this.fullPreview({ mode: true, type: '' });
    }
    this.mdsources!.classList.remove('e-active');
    this.rteObj!.showFullScreen();
});
}

public actionComplete(e: any): void {
    if (e.targetItem === 'Maximize' && isNullOrUndefined(e.args)) {
        this.fullPreview({ mode: true, type: '' });
    } else if (!this.mdSplit!.parentElement!.classList.contains('e-overlay'))
    {
        if (e.targetItem === 'Minimize') {
            this.textArea!.style.display = 'block';
            this.textArea!.style.width = '100%';
            if (this.htmlPreview) { this.htmlPreview.style.display = 'none';
        }

        this.mdSplit!.classList.remove('e-active');
        this.mdsources!.classList.remove('e-active');
    }
    this.markDownConversion();
}

}

public markDownConversion(): void {
    if (this.mdSplit!.classList.contains('e-active')) {
        let id: string = this.rteObj!.getID() + 'html-preview';
        let htmlPreview: HTMLElement = this.rteObj!.element.querySelector('#'
+ id) as HTMLElement;
        htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as
any).getEditPanel() as HTMLTextAreaElement).value);
    }
}

public fullPreview(e: { [key: string]: string | boolean }): void {
    let id: string = this.rteObj!.getID() + 'html-preview';
    this.htmlPreview = this.rteObj!.element.querySelector('#' + id) as
HTMLElement;
    if ((this.mdsources!.classList.contains('e-active') ||
this.mdSplit!.classList.contains('e-active')) && e['mode']) {
        this.mdsources!.classList.remove('e-active');
        this.mdSplit!.classList.remove('e-active');
    }
}

```

```

        this.textArea!.style.display = 'block';
        this.textArea!.style.width = '100%';
        this.htmlPreview.style.display = 'none';
    } else {
        this.mdsource!.classList.add('e-active');
        this.mdSplit!.classList.add('e-active');
        if (!this.htmlPreview) {
            this.htmlPreview = createElement('div', { className: 'e-content'
        });
            this.htmlPreview.id = id;
            this.textArea!.parentNode!.appendChild(this.htmlPreview);
        }
        if (e['type'] === 'preview') {
            this.textArea!.style.display = 'none';
            this.htmlPreview.classList.add('e-pre-source');
        } else {
            this.htmlPreview.classList.remove('e-pre-source');
            this.textArea!.style.width = '50%';
        }
        this.htmlPreview.style.display = 'block';
        this.htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as
any).getEditPanel() as HTMLTextAreaElement).value);
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Custom formation

The Rich Text Editor allows you to customize the markdown syntax by overriding its default syntax. Configure the customized markdown syntax using the [formatter](#) property.

This sample demonstrates how to customize tags of markdown formatting.

For example, apply '+' to Unordered list, apply '1., 2., 3.' to Ordered list, for bold, '[Link to the Video](#)', and for italic ' _ '.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor Markdown Preview Sample
 */
import { Component, ViewChild } from '@angular/core';
import { RichTextEditorComponent, MarkdownFormatter, ToolbarService } from
 '@syncfusion/ej2-angular-richtexteditor';

```

```

import { LinkService, ImageService, MarkdownEditorService } from
 '@syncfusion/ej2-angular-richtexteditor';
import { createElement, KeyboardEventArgs } from '@syncfusion/ej2-base';
import * as Marked from 'marked';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='mdCustom' #mdCustom
[toolbarSettings]='tools' [editorMode]='mode' [formatter]='formatter'
(created)='onCreate()'>
    <ng-template #valueTemplate>
      The sample is configured with customized markdown syntax using the
      __formatter__ property.
      Type the content and click the toolbar item to view customized
      markdown syntax.
      For unordered list, you need to add a plus sign before the word
      (e.g., + list1).
      Or To make a phrase bold, you need to add two underscores before
      and after the phrase (e.g., __this text is bold__).
    </ng-template>
  </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService,
MarkdownEditorService]
})
export class AppComponent {
  @ViewChild('mdCustom')
  public rteObj?: RichTextEditorComponent;
  public textArea?: HTMLTextAreaElement;
  public mdsource?: HTMLElement;
  public tools: object = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|',
      {
        tooltipText: 'Preview',
        template: '<button id="preview-code" class="e-tbar-btn e-control
e-btn e-icon-btn">' +
          '<span class="e-btn-icon e-icons e-md-
preview"></span></button>'
      }, 'Undo', 'Redo']
  };
  public mode: string = 'Markdown';
  public formatter: MarkdownFormatter = new MarkdownFormatter({
    listTags: { 'OL': '1., 2., 3.', 'UL': '+ ' },
    formatTags: {
      'Blockquote': '> '
    },
    selectionTags: { 'Bold': '__', 'Italic': '_' }
  });
  public onCreate(): void {
    this.textArea = (this.rteObj!.contentModule as any).getEditPanel() as
HTMLTextAreaElement;
    this.textArea.addEventListener('keyup', () => {

```

```

        this.markDownConversion();
    });
    this.mdsourse = document.getElementById('preview-code') as any;
    this.mdsourse?.addEventListener('click', (e: MouseEvent) => {
        this.fullPreview();
    });
}
public markDownConversion(): void {
    if (this.mdsourse?.classList.contains('e-active')) {
        let id: string = this.rteObj?.getID() + 'html-view';
        let htmlPreview: Element = this.rteObj!.element.querySelector('#' + id) as Element;
        htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as any).getEditPanel() as HTMLTextAreaElement).value);
    }
}
public fullPreview(): void {
    let id: string = this.rteObj!.getID() + 'html-preview';
    let htmlPreview: HTMLElement = this.rteObj!.element.querySelector('#' + id) as HTMLElement;
    if (this.mdsourse!.classList.contains('e-active')) {
        this.mdsourse!.classList.remove('e-active');
        this.textArea!.style.display = 'block';
        htmlPreview.style.display = 'none';
    } else {
        this.mdsourse!.classList.add('e-active');
        if (!htmlPreview) {
            htmlPreview = createElement('div', { className: 'e-content e-pre-source' });
            htmlPreview.id = id;
            this.textArea!.parentNode!.appendChild(htmlPreview);
        }
        this.textArea!.style.display = 'none';
        htmlPreview.style.display = 'block';
        htmlPreview.innerHTML = Marked(((this.rteObj!.contentModule as any).getEditPanel() as HTMLTextAreaElement).value);
        this.mdsourse!.parentElement!.title = 'Code View';
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

File browser in Angular Rich text editor component

Rich Text Editor allows to browse and insert images in the edit panel using the file browser. File browser allows the users to browse and select a file or folder from the file system and it supports various cloud services.

To get start quickly about Inserting Images in the Angular Rich Text Editor Using a File Manager, refer to the video below.

Required additional dependency

The following list of additional dependencies are required to use the file browser feature in the Rich Text Editor.

```
`js
|-- @syncfusion/ej2-angular-richtexteditor
|-- @syncfusion/ej2-layouts
|-- @syncfusion/ej2-grids
|-- @syncfusion/ej2-filemanager
`,`
```

Additional CSS Reference

Additionally add below styles in the `[src/styles.css]` file.

```
`css

@import "https://ej2.syncfusion.com/angular/documentation/node_modules/@syncfusion/ej2-
layouts/styles/material.css";

@import "../node_modules/@syncfusion/ej2-grids/styles/material.css";

@import "../node_modules/@syncfusion/ej2-filemanager/styles/material.css";
`,`
```

The following example explains about how to configure the file browser within the Rich Text Editor.

- Configure the `FileManager` toolbar item in the `toolbarSettings` API `items` property.
- Set the `enable` property as `true` on `fileManagerSettings` property to make the file browser in the Rich Text Editor to appear on the `FileManager` toolbar click action.

Rich Text Editor features are segregated into individual feature-wise modules. To use the file browser tool, configure `FileManagerService` in providers.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
/**
 * RTE File Browser Sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, QuickToolbarService, FileManagerService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
```

```

selector: 'app-root',
template: `<ejs-richtexteditor [toolbarSettings]='toolbarSettings'
[fileManagerSettings]='fileManagerSettings'>
  <ng-template #valueTemplate>
    <p>The Rich Text Editor is WYSIWYG ("what you see is what you
get") editor useful to create and edit content, and return the valid <a
href="https://ej2.syncfusion.com/home/" target="_blank">HTML markup</a> or <a
href="https://ej2.syncfusion.com/home/" target="_blank">markdown</a> of the
content</p>
    <p><b>Key features:</b></p>
    <ul>
      <li><p>Provides &#60;IFRAME&#62; and &#60;DIV&#62;
modes</p></li>
      <li><p>Capable of handling markdown editing.</p></li>
      <li><p>Contains a modular library to load the necessary
functionality on demand.</p></li>
      <li><p>Provides a fully customizable toolbar.</p></li>
      <li><p>Provides HTML view to edit the source directly for
developers.</p></li>
      <li><p>Supports third-party library integration.</p></li>
      <li><p>Allows preview of modified content before saving
it.</p></li>
      <li><p>Handles images, hyperlinks, video, hyperlinks,
uploads, etc.</p></li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService, FileManagerService]
})
export class AppComponent {
  private hostUrl: string = 'https://ej2-aspcore-
service.azurewebsites.net/';
  public toolbarSettings: Object = {
    items: ['FileManager']
  };
  public fileManagerSettings: object = {
    enable: true,
    path: '/Pictures/Food',
    ajaxSettings: {
      url: this.hostUrl + 'api/FileManager/FileOperations',
      getImageUrl: this.hostUrl + 'api/FileManager/GetImage',
      uploadUrl: this.hostUrl + 'api/FileManager/Upload',
      downloadUrl: this.hostUrl + 'api/FileManager/Download'
    }
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Format Painter in Angular Rich Text Editor Component | Syncfusion

A format painter is a tool that allows you to copy the formatting from a piece of text and apply it to another one. Format Painter can be accessed via the toolbar or the keyboard shortcuts. The format painter can copy the formatting of a single word or a whole paragraph. The format painter can be customized using the [formatPainterSettings](#) property.

Enabling the toolbar option for Format Painter

You can add the **FormatPainter** tool in the Rich Text Editor using the **toolbarSettings** [items](#) property.

Rich Text Editor features are segregated into individual feature-wise modules. To use the Format Painter feature, we need to import and inject the **FormatPainterService** in the **@NgModule.providers** section.

By double-clicking the format painter toolbar button, **sticky mode** will be enabled. In sticky mode, the format painter will be disabled when the user clicks the **Escape** key again.

The following code example shows how to add the format painter tool in the Rich Text Editor.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService, FormatPainterService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor [toolbarSettings]='toolbarSettings'>
    <ng-template #valueTemplate>
      <h3><strong>Format Painter</strong></h3>
      <p>
        A Format Painter is a Rich Text Editor feature
        allowing users to quickly
        <span style="background-color: rgb(198, 140, 83);"><strong>copy</strong></span>
        and
        <span style="background-color: rgb(198, 140, 83);"><strong>paste</strong></span>
        formatting from one text to another. With a rich text
        editor, utilize the format painter as follows:
      </p>
      <ul>
        <li>
          Select the text whose format you want to copy.
        </li>
        <li>
```



```

Click on the <strong><em>Format
Painter</em></strong> button in the toolbar. It may look like a paintbrush
icon.
    </li>
    <li>
        The cursor will change to a
        <strong>paintbrush</strong> icon. Click and drag the cursor over the text you
        want to apply the copied format.
    </li>
    <li>
        Release the mouse button to apply the format.
    </li>
</ul>
<p>
    Using the format painter in a rich text editor can
    save you time when formatting a large document, You can quickly
    copy and apply formatting
    to <span style="background-color: rgb(198, 140,
83);"><strong>multiple sections</strong></span>.
    It's a helpful tool for anyone who works with text
    editing regularly, such as writers, editors, and content creators.
</p>
</ng-template>
</ejs-richtexteditor>`,
    providers: [
        ToolbarService, LinkService, ImageService, HtmlEditorService,
        QuickToolbarService, FormatPainterService
    ],
} )
export class AppComponent {
    public toolbarSettings: object = {
        items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic',
'Underline', '|', 'Formats', 'Alignments',
'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
'SourceCode', 'Undo', 'Redo']
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customization of copy and paste format

You can customize the format painter tool in the Rich Text Editor using the `formatPainterSettings` property.

The `allowedFormats` property helps you to specify tag names that allow the formats to be copied from the selected text. For instance, you can include formats from the selected text using tags like `p`; `h1`; `h2`; `h3`; `div`; `ul`; `ol`; `li`; `span`; `strong`; `em`; `code`; . The following example demonstrates how to customize this functionality.

Similarly, with the [deniedFormats](#) property, you can utilize the selectors to prevent specific formats from being pasted onto the selected text. The table below illustrates the selectors and their respective usage.

Type	Description	Selector	Usage
	-----	-----	-----
()	Class Selector	h3(e-rte-block-blue-text)	The class name e-rte-block-blue-text of H3 element is not copied.
[]	Attribute Selector	span\[title]	The title attribute of span element is not copied.
{}	Style Selector	span{background-color, color}	The background-color and color styles of span element is not copied.

Using the `deniedFormats` property following styles are denied copying from the selected text such as `h3(e-rte-block-blue-text){background-color,padding}[title]; li{color}; span(e-inline-text-highlight)[title]; strong{color}(e-rte-strong-bg).`

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService, FormatPainterService, FormatPainterSettingsModel } from
 '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor [toolbarSettings]='toolbarSettings'
[formatPainterSettings]='formatPainterSettings'>
    <ng-template #valueTemplate>
      <h3 class="e-rte-block-blue-text" title="Format Painter"
style="color: #0079f3; background-color: #eff6ff; padding: 10px;"><strong
>Format Painter</strong></h3>
      <p>
        A Format Painter is a Rich Text Editor feature
        allowing users to quickly
        <span class="e-inline-text-highlight" style="color:
blue;" title="Styled by CSS Class selector"><strong>copy</strong></span>
        and
        <span class="e-inline-text-highlight" style="color:
blue;" title="Styled by CSS Class selector"><strong>paste</strong></span>
        formatting from one text to another. With a rich text
        editor, utilize the format painter as follows:
      </p>
    </ng-template>
  `
})
```

```

        </p>
        <ul>
            <li style="color: crimson;">
                Select the text whose format you want to copy.
            </li>
            <li style="color: crimson;">
                Click on the <strong><em>Format
Painter</em></strong> button in the toolbar. It may look like a paintbrush
icon.
            </li>
            <li style="color: crimson;">
                The cursor will change to a
<strong>paintbrush</strong> icon. Click and drag the cursor over the text you
want to apply the copied format.
            </li>
            <li style="color: crimson;">
                Release the mouse button to apply the format.
            </li>
        </ul>
        <p>
            Using the format painter in a rich text editor can
save you time when formatting a large document, You can quickly
copy and apply formatting
to <strong class="e-rte-strong-bg" style="color:
blue">multiple sections</strong>.
            It's a helpful tool for anyone who works with text
editing regularly, such as writers, editors, and content creators.
        </p>
    </ng-template>
</ejs-richtexteditor>`,
    providers: [
        ToolbarService, LinkService, ImageService, HtmlEditorService,
        QuickToolbarService, FormatPainterService
    ],
} )
export class AppComponent {
    public toolbarSettings: object = {
        items: ['FormatPainter', 'ClearFormat', 'Bold', 'Italic',
'Underline', '|', 'Formats', 'Alignments',
'OrderedList', 'UnorderedList', '|', 'CreateLink', 'Image', '|',
'SourceCode', 'Undo', 'Redo']
    }
    public formatPainterSettings: FormatPainterSettingsModel = {
        allowedFormats: 'p;h1;h2;h3;div;ul;ol;li;span;strong;em;code;',
        deniedFormats: 'h3(e-rte-block-blue-text){background-
color,padding,color}[title]; li{color}; span(e-inline-text-
highlight){color}[title]; strong{color}(e-rte-strong-bg);',
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Using the shortcut key to copy and paste the format

You can use the following shortcut keys to copy and paste the format in the Rich Text Editor.

Actions	Keyboard shortcuts	Description
Copy the format	Alt + Shift + c	Copy the selection format or current range.
Paste the format	Alt + Shift + v	Paste the copied format.
Escape	Esc	Remove the previously copied format and disable the sticky mode.

The format painter retains the formatting after application making it possible to apply the same formatting multiple times by using the Alt + Shift + v keyboard shortcut.

Additionally, You can perform the format painter actions programmatically using the [executeCommand](#) public method.

Validation in Angular Rich text editor component

The Rich Text Editor supports both the reactive and template-driven form-building technologies.

Template driven forms

The template-driven forms use the `angular` directives in view to handle the forms controls. To enable the template-driven, import the FormsModule into corresponding app component.

For more details about template-driven

forms, refer to: <https://angular.io/guide/forms#template-driven-forms>.

Mention the `name` attribute to Rich Text Editor element that can be used to identify the form element. To register a Rich Text Editor element to ngForm, give the ngModel to it. So, the FormsModule will automatically detect the Rich Text Editor as a form element. After that, the Rich Text Editor value will be selected based on the ngModel value.

The following example demonstrates how to achieve a two-way data binding.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    FormsModule,
    ButtonModule,
```

```

    ReactiveFormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <div class="content-wrapper">
      <div id='content' class='box-form' style="margin: 0 auto; max-
width:750px; padding:25px">
        <form (ngSubmit)="onSubmit()" #rteForm="ngForm">
          <div class="form-group">
            <ejs-richtexteditor #fromRTE #name='ngModel'
[(value)]='value' required name="name"
[(ngModel)]="value" (created)="rteCreated()"></ejs-
richtexteditor>
            <div *ngIf="(name.invalid && name.touched)" class="alert
alert-danger">
              <div *ngIf="name.errors![ 'required' ]">
                Value is required.
              </div>
            </div>
          </div>
          <div>
            <button type="submit" ejs-button
[disabled]="!rteForm.valid">Submit</button>
            <button type="reset" ejs-button style="margin-left:
20px">Reset</button>
          </div>
        </form>
      </div>
    </div>
  </div>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public value: string | null = null;
  @ViewChild('fromRTE')
  private rteEle: RichTextEditorComponent | undefined;
  rteCreated(): void {
    this.rteEle!.element.focus();
  }
  onSubmit(): void {
    alert('Form submitted successfully');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Reactive forms

The reactive forms use the reactive model-driven technique to handle form data between the component and view. So, it is called as **model-driven** forms. It listens the form data changes between App component and view also returns the valid states and values of form elements.

For more details about Reactive Forms, refer to: <https://angular.io/guide/reactive-forms>.

For the reactive forms, import a **ReactiveFormsModule** into app module as well as the **FormGroup**, **FormControl** should be imported to app component. The **FormGroup** is used to declare **formGroupName** for the form group and the **FormControl** is used to declare **formControlName** for form controls. You can declare the **formControlName** to Rich Text Editor a value object must be created to the **FormGroup** and each value will be the default value of the form control.

The following example demonstrates how to use the reactive forms.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
import { FormBuilder, FormControl, FormGroup, Validators } from
'@angular/forms';
@Component({
  imports: [

    RichTextEditorAllModule,
    FormsModule,
    ButtonModule,
    ReactiveFormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">    <div class="content-wrapper">
<div id="content" class="box-form" style="margin: 0 auto; max-width:750px;
padding:25px">        <form [formGroup]="rteForm"
  (ngSubmit)="onSubmit()">            <div class="form-group">
<ejs-richtexteditor #fromRTE formControlName="name" (created)="rteCreated()">
</ejs-richtexteditor>                <div *ngIf="rteForm!.invalid &&
rteForm!.controls['name'].touched" class="alert alert-danger">
<div *ngIf="rteForm.controls['name'].errors!['required']">
Value is required.                </div>                </div>
</div>                <div class="form-group">                <button
type="submit" ejs-button [disabled]="!rteForm.valid">Submit</button>
<button type="reset" ejs-button (click)="rteForm.reset()" style="margin-left:
20px">Reset</button>                </div>                </form>                </div>
</div></div>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  rteForm!: FormGroup;
```

```

@ViewChild('fromRTE')
private rteEle: RichTextEditorComponent | undefined;
constructor() {
  // <--- inject FormBuilder
}
ngOnInit(): void {
  this.rteForm = new FormGroup({
    'name': new FormControl(null, Validators.required)
  });
}
rteCreated(): void {
  this.rteEle!.element.focus();
}
onSubmit(): void {
  alert('Form submitted successfully');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Globalization in Angular Rich text editor component

Localization

The Rich Text Editor provides an option to localize its strings. It is used to adapting the editor to a particular local language. By default, the editor will use the US English (en-US) as its language. Please find the table with a list of keys and their corresponding values for the default language (en-US).

`typescript

'en-US': {

'richtexteditor': {

alignments: "Alignments",

justifyLeft: "Align Left",

justifyCenter: "Align Center",

justifyRight: "Align Right",

justifyFull: "Align Justify",

fontName: "Font Name",

fontSize: "Font Size",

fontColor: "Font Color",

backgroundColor: "Background Color",

bold: "Bold",

italic: "Italic",

underline: "Underline",
strikethrough: "Strikethrough",
clearFormat: "Clear Format",
clearAll: "Clear All",
cut: "Cut",
copy: "Copy",
paste: "Paste",
unorderedList: "Bulleted List",
orderedList: "Numbered List",
indent: "Increase Indent",
outdent: "Decrease Indent",
undo: "Undo",
redo: "Redo",
superscript: "Superscript",
subscript: "Subscript",
createLink: "Insert Link",
openLink: "Open Link",
editLink: "Edit Link",
removeLink: "Remove Link",
image: "Insert Image",
replace: "Replace",
align: "Align",
caption: "Image Caption",
remove: "Remove",
insertLink: "Insert Link",
display: "Display",
altText: "Alternative Text",
dimension: "Change Size",
fullscreen: "Maximize",
maximize: "Maximize",
minimize: "Minimize",
lowerCase: "Lower Case",
upperCase: "Upper Case",


```
print: "Print",
formats: "Formats",
sourcecode: "Code View",
preview: "Preview",
viewside: "ViewSide",
insertCode: "Insert Code",
linkText: "Display Text",
linkTooltipLabel: "Title",
linkWebUrl: "Web Address",
linkTitle: "Enter a title",
linkurl: "http://example.com",
linkOpenInNewWindow: "Open Link in New Window",
linkHeader: "Insert Link",
dialogInsert: "Insert",
dialogCancel: "Cancel",
dialogUpdate: "Update",
imageHeader: "Insert Image",
imageLinkHeader: "You can also provide a link from the web",
mdimageLink: "Please provide a URL for your image",
imageUploadMessage: "Drop image here or browse to upload",
imageDeviceUploadMessage: "Click here to upload",
imageAlternateText: "Alternate Text",
alternateHeader: "Alternative Text",
browse: "Browse",
imageUrl: "http://example.com/image.png",
imageCaption: "Caption",
imageSizeHeader: "Image Size",
imageHeight: "Height",
imageWidth: "Width",
textPlaceholder: "Enter Text",
inserttablebtn: "Insert Table",
tabledialogHeader: "Insert Table",
tableWidth: "Width",
```

cellpadding: "Cell Padding",
cellspacing: "Cell Spacing",
columns: "Number of columns",
rows: "Number of rows",
tableRows: "Table Rows",
tableColumns: "Table Columns",
tableCellHorizontalAlign: "Table Cell Horizontal Align",
tableCellVerticalAlign: "Table Cell Vertical Align",
createTable: "Create Table",
removeTable: "Remove Table",
tableHeader: "Table Header",
tableRemove: "Table Remove",
tableCellBackground: "Table Cell Background",
tableEditProperties: "Table Edit Properties",
styles: "Styles",
insertColumnLeft: "Insert Column Left",
insertColumnRight: "Insert Column Right",
deleteColumn: "Delete Column",
insertRowBefore: "Insert Row Before",
insertRowAfter: "Insert Row After",
deleteRow: "Delete Row",
tableEditHeader: "Edit Table",
TableHeadingText: "Heading",
TableColText: "Col",
imageInsertLinkHeader: "Insert Link",
editImageHeader: "Edit Image",
alignmentsDropDownLeft: "Align Left",
alignmentsDropDownCenter: "Align Center",
alignmentsDropDownRight: "Align Right",
alignmentsDropDownJustify: "Align Justify",
imageDisplayDropDownInline: "Inline",
imageDisplayDropDownBreak: "Break",
tableInsertRowDropDownBefore: "Insert row before",

tableInsertRowDropDownAfter: "Insert row after",
tableInsertRowDropDownDelete: "Delete row",
tableInsertColumnDropDownLeft: "Insert column left",
tableInsertColumnDropDownRight: "Insert column right",
tableInsertColumnDropDownDelete: "Delete column",
tableVerticalAlignDropDownTop: "Align Top",
tableVerticalAlignDropDownMiddle: "Align Middle",
tableVerticalAlignDropDownBottom: "Align Bottom",
tableStylesDropDownDashedBorder: "Dashed Borders",
tableStylesDropDownAlternateRows: "Alternate Rows",
pasteFormat: "Paste Format",
pasteFormatContent: "Choose the formatting action",
plainText: "Plain Text",
cleanFormat: "Clean",
keepFormat: "Keep",
formatsDropDownParagraph: "Paragraph",
formatsDropDownCode: "Code",
formatsDropDownQuotation: "Quotation",
formatsDropDownHeading1: "Heading 1",
formatsDropDownHeading2: "Heading 2",
formatsDropDownHeading3: "Heading 3",
formatsDropDownHeading4: "Heading 4",
fontNameSegoeUI: "Segoe UI",
fontNameArial: "Arial",
fontNameGeorgia: "Georgia",
fontNameImpact: "Impact",
fontNameTahoma: "Tahoma",
fontNameTimesNewRoman: "Times New Roman",
fontNameVerdana: "Verdana",
numberFormatListNumber: 'Number',
numberFormatListLowerAlpha: 'LowerAlpha',
numberFormatListUpperAlpha: 'UpperAlpha',
numberFormatListLowerRoman: 'LowerRoman',

```

numberFormatListUpperRoman: 'UpperRoman',
numberFormatListLowerGreek: 'LowerGreek',
bulletFormatListDisc: 'Disc',
bulletFormatListCircle: 'Circle',
bulletFormatListSquare: 'Square',
numberFormatListNone: 'None',
bulletFormatListNone: 'None',
formatPainter: 'Format Painter',
emojiPicker: 'Emoji Picker',
embeddedCode: 'Embedded Code',
pasteEmbeddedCodeHere: 'Paste Embedded Code here',
emojiPickerTypeToFind: 'Type to find',
emojiPickerNoResultFound: 'No results found',
emojiPickerTrySomethingElse: 'Try something else',
}
}
`

```

To localize the editor's strings with your own localization, copy the default language informations and localize the strings in the values column. For example, to localize the editor in German language ("de-DE").

```

`typescript
'de-DE': {
  'richtexteditor': {
    alignments: "Alignments",
    justifyLeft: "Ausrichten von Text links",
    justifyCenter: "Text-Zentrum",
    justifyRight: "Ausrichten von Text rechts",
    justifyFull: "rechtfertigen",
    fontName: "Wählen Sie Schriftfamilie",
    fontSize: "Wählen Sie Schriftgröße",
    fontColor: "Wählen Sie die Farbe",
    backgroundColor: "Hintergrundfarbe",
    bold: "fett",
    italic: "kursiv",

```

underline: "unterstreichen",
strikethrough: "Durchgestrichen",
clearAll: "Alles",
clearFormat: "Klar Format",
cut: "schneiden",
copy: "Kopieren",
paste: "Paste",
unorderedList: "Legen Sie ungeordnete Liste",
orderedList: "Geordnete Liste einfügen",
indent: "Einzug",
outdent: "Einzug verkleinern",
undo: "lösen",
redo: "Wiederherstellen",
superscript: "Überschrift",
subscript: "index",
createLink: "Link einfügen",
removeLink: "fjern Hyperlink",
openLink: "Open link",
editLink: "Edit link",
image: "Bild einfügen",
replace: "ersetzen",
align: "ausrichten",
caption: "Bildbeschriftung",
formats: "Formats",
remove: "Löschen",
insertLink: "Link einfügen",
display: "Anzeige",
alttext: "alternativer Text",
dimension: "Größe",
fullscreen: "Vollbild",
maximize: "Maximieren",
minimize: "minimieren",
zoomIn: "hineinzoomen",

zoomOut: "Rauszoomen",
upperCase: "Großbuchstaben",
lowerCase: "Kleinbuchstaben",
print: "Drucken",
sourcecode: "Quellcode",
preview: "Vorschau",
viewside: "Seite anzeigen",
insertcode: "Code eingeben",
linkText: "Displaytext",
linkTooltipLabel: "tooltip",
linkWebUrl: "Webadres",
linkOpenInNewWindow: "Open de link in een nieuw venster",
linkHeader: "Link invoegen",
dialogInsert: "invoegen",
dialogCancel: "Annuleer",
dialogUpdate: "Bijwerken",
imageHeader: "Voeg afbeelding in",
imageLinkHeader: "U kunt ook een link van internet opgeven",
imageUploadMessage: "Zet hier een afbeelding neer of klik om te uploaden",
imageDeviceUploadMessage: "Klik hier om te uploaden",
imageAlternateText: "Alternatieve tekst",
alternateHeader: "Alternatieve tekst",
browse: "Blader",
imageUrl: "URL",
imageCaption: "onderschrift",
imageSizeHeader: "Afbeeldingsgrootte",
imageHeight: "Hoogte",
imageWidth: "Breedte",
textPlaceholder: "Text eingeben",
inserttablebtn: "Tabelle einfügen",
tabledialogHeader: "Tabelle einfügen",
tableWidth: "Breite",
cellpadding: "Zellauffüllung",

cellspacing: "Zellabstand",
columns: "Anzahl der Spalten",
rows: "Reihenanzahl",
tableRows: "Tabellenzeilen",
tableColumns: "Tabellenspalten",
tableCellHorizontalAlign: "Horizontale Ausrichtung der Tabellenzelle",
tableCellVerticalAlign: "Vertikale Ausrichtung der Tabellenzelle",
createTable: "Tabelle erstellen",
removeTable: "Tabelle entfernen",
tableHeader: "Tabellenkopfzeile",
tableRemove: "Tabelle entfernen",
tableCellBackground: "Tabellenzellenhintergrund",
tableEditProperties: "Eigenschaften der Tabellenbearbeitung",
styles: "Styles",
insertColumnLeft: "Spalte links einfügen",
insertColumnRight: "Spalte rechts einfügen",
deleteColumn: "Spalte löschen",
insertRowBefore: "Zeile vor einfügen",
insertRowAfter: "Zeile einfügen nach",
deleteRow: "Zeile löschen",
tableEditHeader: "Tabelle bearbeiten",
TableHeadingText: "Überschrift",
TableColText: "Col",
imageInsertLinkHeader: "Link einfügen",
editImageHeader: "Bild bearbeiten",
alignmentsDropDownLeft: "Linksbündig",
alignmentsDropDownCenter: "Im Zentrum anordnen",
alignmentsDropDownRight: "Rechts ausrichten",
alignmentsDropDownJustify: "Justize ausrichten",
imageDisplayDropDownInline: "In der Reihe",
imageDisplayDropDownBreak: "Brechen",
tableInsertRowDropDownBefore: "Reihe vorher einfügen",
tableInsertRowDropDownAfter: "Zeile danach einfügen",

```

tableInsertRowDropDownDelete: "Zeile löschen",
tableInsertColumnDropDownLeft: "Spalte links einfügen",
tableInsertColumnDropDownRight: "Spalte rechts einfügen",
tableInsertColumnDropDownDelete: "Spalte löschen",
tableVerticalAlignDropDownTop: "Top ausrichten",
tableVerticalAlignDropDownMiddle: "Mitte ausrichten",
tableVerticalAlignDropDownBottom: "Unten ausrichten",
tableStylesDropDownDashedBorder: "Gestrichelte Grenzen",
tableStylesDropDownAlternateRows: "Alternative Zeilen",
pasteFormat: "Format einfügen",
pasteFormatContent: "Wählen Sie die Formatierungsaktion aus",
plainText: "Einfacher Text",
cleanFormat: "sauber",
keepFormat: "Behalten",
formatsDropDownParagraph: "Absatz",
formatsDropDownCode: "Kodex",
formatsDropDownQuotation: "Zitat",
formatsDropDownHeading1: "Überschrift 1",
formatsDropDownHeading2: "Überschrift 2",
formatsDropDownHeading3: "Überschrift 3",
formatsDropDownHeading4: "Überschrift 4",
fontNameSegoeUI: "Segoe UI",
fontNameArial: "Arial",
fontNameGeorgia: "Georgia",
fontNameImpact: "Einschlag",
fontNameTahoma: "Tahoma",
fontNameTimesNewRoman: "Mal Neu römisch",
fontNameVerdana: "Verdana"
}
},
`

```

The below sample demonstrate that, the Rich Text Editor control rendered with 'de-DE' German language using [locale](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { enableRipple } from '@syncfusion/ej2-base';
import { L10n } from '@syncfusion/ej2-base';
enableRipple(true);
/**
 * Rich Text Editor localization sample
 */
L10n.load({
  'de-DE': {
    'richtexteditor': {
      alignments: "Ausrichtungen",
      justifyLeft: "Linksbündig",
      justifyCenter: "Im Zentrum anordnen",
      justifyRight: "Rechts ausrichten",
      justifyFull: "Justize ausrichten",
      fontName: "Schriftartenname",
      fontSize: "Schriftgröße",
      fontColor: "Schriftfarbe",
      backgroundColor: "Hintergrundfarbe",
      bold: "Fett gedruckt",
      italic: "Kursiv",
      underline: "Unterstreichen",
      strikethrough: "Durchgestrichen",
      clearFormat: "Format löschen",
      clearAll: "Alles löschen",
      cut: "Schnitt",
      copy: "Kopieren",
      paste: "Einfügen",
      unorderedList: "Liste mit Aufzählungszeichen",
      orderedList: "Nummerierte Liste",
      indent: "Einzug vergrößern",
      outdent: "Einzug verringern",
      undo: "Rückgängig machen",
      redo: "Wiederholen",
      superscript: "Hochgestellt",
      subscript: "Index",
      createLink: "Link einfügen",
      openLink: "Verbindung öffnen",
      editLink: "Link bearbeiten",
      removeLink: "Link entfernen",
      image: "Bild einfügen",
      replace: "Ersetzen",
      align: "Ausrichten",
      caption: "Bildbeschreibung",
      remove: "Löschen",
      insertLink: "Link einfügen",
      display: "Anzeige",
      altText: "alternativer Text",
      dimension: "Größe ändern",
      fullscreen: "Maximieren",
      maximize: "Maximieren",

```

```

minimize: "Minimieren",
lowerCase: "Kleinbuchstaben",
upperCase: "Großbuchstaben",
print: "Drucken",
formats: "Formate",
sourcecode: "Codeansicht",
preview: "Vorschau",
viewside: "Ansicht Seite",
insertCode: "Code eingeben",
linkText: "Text anzeigen",
linkTooltipLabel: "Titel",
linkWebUrl: "Webadresse",
linkTitle: "Titel eingeben",
linkurl: "http://example.com",
linkOpenInNewWindow: "Link in neuem Fenster öffnen",
linkHeader: "Link einfügen",
dialogInsert: "Einfügen",
dialogCancel: "Stornieren",
dialogUpdate: "Aktualisieren",
imageHeader: "Bild einfügen",
imageLinkHeader: "Sie können auch einen Link aus dem Web
bereitstellen",
mdimageLink: "Bitte geben Sie eine URL für Ihr Bild an",
imageUploadMessage: "Bild hier ablegen oder hoch laden",
imageDeviceUploadMessage: "Klicken Sie hier zum Hochladen",
imageAlternateText: "Alternativer Text",
alternateHeader: "alternativer Text",
browse: "Durchsuche",
imageUrl: "http://example.com/image.png",
imageCaption: "Bildbeschriftung",
imageSizeHeader: "Bildgröße",
imageHeight: "Höhe",
imageWidth: "Breite",
textPlaceholder: "Text eingeben",
inserttablebtn: "Tabelle einfügen",
tabledialogHeader: "Tabelle einfügen",
tableWidth: "Breite",
cellpadding: "Zellauffüllung",
cellspacing: "Zellabstand",
columns: "Anzahl der Spalten",
rows: "Reihenanzahl",
tableRows: "Tabellenzeilen",
tableColumns: "Tabellenspalten",
tableCellHorizontalAlign: "Horizontale Ausrichtung der
Tabellenzelle",
tableCellVerticalAlign: "Vertikale Ausrichtung der
Tabellenzelle",
createTable: "Tabelle erstellen",
removeTable: "Tabelle entfernen",
tableHeader: "Tabellenkopfzeile",
tableRemove: "Tabelle entfernen",
tableCellBackground: "Tabellenzellenhintergrund",
tableEditProperties: "Eigenschaften der Tabellenbearbeitung",
styles: "Styles",
insertColumnLeft: "Spalte links einfügen",
insertColumnRight: "Spalte rechts einfügen",
deleteColumn: "Spalte löschen",

```

```

insertRowBefore: "Zeile vor einfügen",
insertRowAfter: "Zeile einfügen nach",
deleteRow: "Zeile löschen",
tableEditHeader: "Tabelle bearbeiten",
TableHeadingText: "Überschrift",
TableColText: "Col",
imageInsertLinkHeader: "Link einfügen",
editImageHeader: "Bild bearbeiten",
alignmentsDropDownLeft: "Linksbündig",
alignmentsDropDownCenter: "Im Zentrum anordnen",
alignmentsDropDownRight: "Rechts ausrichten",
alignmentsDropDownJustify: "Justize ausrichten",
imageDisplayDropDownInline: "In der Reihe",
imageDisplayDropDownBreak: "Brechen",
tableInsertRowDropDownBefore: "Reihe vorher einfügen",
tableInsertRowDropDownAfter: "Zeile danach einfügen",
tableInsertRowDropDownDelete: "Zeile löschen",
tableInsertColumnDropDownLeft: "Spalte links einfügen",
tableInsertColumnDropDownRight: "Spalte rechts einfügen",
tableInsertColumnDropDownDelete: "Spalte löschen",
tableVerticalAlignDropDownTop: "Top ausrichten",
tableVerticalAlignDropDownMiddle: "Mitte ausrichten",
tableVerticalAlignDropDownBottom: "Unten ausrichten",
tableStylesDropDownDashedBorder: "Gestrichelte Grenzen",
tableStylesDropDownAlternateRows: "Alternative Zeilen",
formatsDropDownParagraph: "Absatz",
formatsDropDownCode: "Kodex",
formatsDropDownQuotation: "Zitat",
formatsDropDownHeading1: "Überschrift 1",
formatsDropDownHeading2: "Überschrift 2",
formatsDropDownHeading3: "Überschrift 3",
formatsDropDownHeading4: "Überschrift 4",
fontNameSegoeUI: "Segoe UI",
fontNameArial: "Arial",
fontNameGeorgia: "Georgia",
fontNameImpact: "Einschlag",
fontNameTahoma: "Tahoma",
fontNameTimesNewRoman: "Mal Neu römisch",
fontNameVerdana: "Verdana"
    }
  }
});
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService }
from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' locale = 'de-DE'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService,
    HtmlEditorService]

```

```

    })
    export class AppComponent {
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

RTL

Specifies the direction of the Rich Text 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.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [enableRtl] = 'rtl'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public rtl = true;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Miscellaneous in Angular Rich text editor component

Placeholder

Specifies the placeholder for the Rich Text Editor's content used when the Rich Text Editor body is empty through the [placeholder](#) property.

Through the `e-rte-placeholder` class to define our custom font family, font color, and styles to the placeholder text.

`typescript

```
.e-richtexteditor .e-rte-placeholder {
```

```
font-family: monospace;
```

```
}
```

```
,
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' placeholder='Type Something'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Character count

The Rich Text Editor automatically counts the number of characters in the content are while typing using the [showCharCount](#) property. The characters count displayed at the bottom of the editor. You can limit the number of characters in your content using the [maxLength](#) property. By default, the editor sets the characters limit value is infinity.

The character count color will be modified based on the characters in the RichTextEditor.

| Status | Description |

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

| normal | Till 70% of given maxLength count reach, character count color is black. |

| warning | Once the number of character count in the Rich Text Editor reached 70% of given maxLength count, the character count color will be orange, indicating that, the Rich Text Editor value going to reach the maximum count. |

| error | Once the number of character count in the Rich Text Editor reached 90% of given maxLength count, the character count color will be red, indicating that, the Rich Text Editor value reached the maximum count. |

To use quick **Character Count** feature, inject **CountService** in the provider section of **AppModule**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [showCharCount]='true'
[maxLength]='maxLength'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService]
})
export class AppComponent {
  public maxLength = 2000;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Code view

RichTextEditor includes the ability for users to directly edit HTML code via **Source View** in the text area. If you made any modification in Source view directly, the changes will be reflected in the Rich Text Editor's content. So, the users will have more flexibility over the content they have created.

This sample used [Code mirror](#) plugin helps to highlight the HTML content and when changes happens in code view, the same has been reflected in preview mode.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
import { RichTextEditorComponent, CountService } from '@syncfusion/ej2-angular-richtexteditor';
import { createElement } from '@syncfusion/ej2-base';
// import * as CodeMirror from 'codemirror';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #toolsRTE id='alltoolRTE'
[toolbarSettings]='tools'
[showCharCount]='true' (actionComplete)='actionCompleteHandler($event)'
[maxLength]='maxLength'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor is WYSIWYG ("what you see is what you
get") editor useful to create and edit content and return
      the valid <a href="https://ej2.syncfusion.com/home/"
target="_blank">HTML markup</a> or
      <a href="https://ej2.syncfusion.com/home/"
target="_blank">markdown</a> of the content</p>
      <p><b>Toolbar</b></p>
      <ol>
        <li>
          <p>Toolbar contains commands to align the text, insert
link, insert image, insert list, undo/redo operations, HTML view, etc </p>
        </li>
        <li>
          <p>Toolbar is fully customizable </p>
        </li>
      </ol>
      <p><b>Links</b></p>
      <ol>
        <li>
          <p>You can insert a hyperlink with its corresponding
dialog </p>
        </li>
        <li>
          <p>Attach a hyperlink to the displayed text. </p>
        </li>
        <li>
```

```

        <p> Customize the quick toolbar based on the
hyperlink </p>
        </li>
    </ol>
    <p><b>Image.</b></p>
    <ol>
        <li>
            <p> Allows you to insert images from an online source
as well as the
            local computer </p>
        </li>
        <li>
            <p> You can upload an image </p>
        </li>
        <li>
            <p>Provides an option to customize quick toolbar for
an image </p>
        </li>
    </ol>
</ng-template>
</ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService]
})
export class AppComponent implements AfterViewInit {
    @ViewChild('toolsRTE') public rteObj?: RichTextEditorComponent;
    public tools: object = {
        items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };
    public maxLength: number = 1000;
    public textArea?: HTMLElement;
    public myCodeMirror?: any;
    ngAfterViewInit(): void {
        let rteObj: RichTextEditorComponent = this.rteObj as any;
        setTimeout(() => { this.textArea = (rteObj.contentModule as
any).getEditPanel() as HTMLElement; }, 600);
    }
    public mirrorConversion(e?: any): void {
        let id: string = this.rteObj!.getID() + 'mirror-view';
        let mirrorView: HTMLElement = this.rteObj!.element.querySelector('#' +
id) as HTMLElement;
        let charCount: HTMLElement = this.rteObj!.element.querySelector('.e-rte-
character-count') as HTMLElement;
        if (e.targetItem === 'Preview') {
            this.textArea!.style.display = 'block';
            mirrorView.style.display = 'none';
            this.textArea!.innerHTML = this.myCodeMirror.getValue();
            charCount.style.display = 'block';
        } else {
            if (!mirrorView) {
                mirrorView = createElement('div', { className: 'e-content' });

```



```

        mirrorView.id = id;
        this.textArea!.parentNode!.appendChild(mirrorView);
    } else {
        mirrorView.innerHTML = '';
    }
    this.textArea!.style.display = 'none';
    mirrorView.style.display = 'block';
    this.renderCodeMirror(mirrorView, this.rteObj!.value);
    charCount.style.display = 'none';
}
}

public renderCodeMirror(mirrorView: HTMLElement, content: string): void {
    // this.myCodeMirror = CodeMirror(mirrorView, {
    //     value: content,
    //     lineNumbers: true,
    //     mode: 'text/html',
    //     lineWrapping: true,
    // });
}

public actionCompleteHandler(e: any): void {
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
    'Preview')) {
        (this.rteObj!.sourceCodeModule.getPanel() as
    HTMLTextAreaElement).style.display = 'none';
        this.mirrorConversion(e);
    } else {
        setTimeout(() => {
    this.rteObj!.toolbarModule.refreshToolbarOverflow(); }, 400);
        }
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Undo/redo manager

Undo and redo tools allow you to edit the text by disregard/cancel the recently made changes and restore it to previous state. It is a useful tool to restore the performed action which got changed by mistake. By default, upto 30 actions can be undo/redo in the editor.

To undo and redo operations, do one of the following:

- Press the undo/redo button on the toolbar.
- Press the **Ctrl + Z/Ctrl + Y** combination on the keyboard.

Customize the undo/redo step count using the [undoRedoSteps](#) property. By default, undo/redo actions take 300ms time interval for store the action to the undoRedoManager. The time interval can be customized by using the [undoRedoTimer](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'
[undoRedoSteps] ='steps' [undoRedoTimer]='timer'>
    </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService]
})
export class AppComponent {
  public tools = {items: ['Undo', 'Redo']};
  public steps = 50;
  public timer = 400;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Prevention of cross-site scripting (XSS)

The Rich Text Editor allow the users to edit the content with security by preventing cross-site scripting (XSS). By default, provided built-in support to remove the elements from editor content which cause XSS attack. The editor removes the elements based on the attributes if there is possible to execute script.

In the following sample, removed `script` tag and `onmouseover` attribute from content of the Rich Text Editor.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

```

```

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor id='defaultRTE' [value]='rteValue'></ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
  })
  export class AppComponent {
    public rteValue: string = `<div
onmouseover='javascript:alert(1)'>Prevention of Cross Site Scripting (XSS)
</div><script>alert('hi')</script>`;
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

It's only applicable to editorMode as HTML.

Custom cross-site scripting

You can also filter the elements and attributes additionally which cause the XSS attack through [beforeSanitizeHtml](#) event. Return the value from the event argument [helper](#) function to apply in the editor. If you want to prevent the built-in support and make own cross-site scripting rules, set [cancel](#) argument as true.

The following sample demonstrate how to filter [script](#) tag from value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, BeforeSanitizeHtmlArgs } from '@syncfusion/ej2-angular-richtexteditor';
import { detach } from '@syncfusion/ej2-base';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [value]='rteValue'
(beforeSanitizeHtml)='onBeforeSanitizeHtml($event)'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})

```

```

    })
    export class AppComponent {
        public rteValue:string =`<div>Prevention of Cross Sit Scripting
(XSS)</div><script>alert('hi')</script>`;
        public onBeforeSanitizeHtml(e: BeforeSanitizeHtmlArgs): void {
            e.helper = (value: string) => {
                e.cancel = true;
                let temp: HTMLElement = document.createElement('div');
                temp.innerHTML = value;
                let scriptTag: HTMLElement = temp.querySelector('script') as
                HTMLElement;
                if (scriptTag) {
                    detach(scriptTag);
                }
                return temp.innerHTML;
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Resizable support

This feature allows the editor to be resized dynamically. The users can enable or disable this feature using the `enableResize` property in the Rich Text Editor. If `enableResize` is set to true, the Rich Text Editor component creates grip at the bottom right corner, which allows resizing the component in the diagonal direction. The following sample demonstrates the resizable feature.

Enabling the resizable support

To render the Rich Text Editor in the resizable mode, set the `enableResize` property to true. The above feature is segregated into individual feature-wise module. To use Resizable feature, inject `ResizeService` in the provider section of `AppModule`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, ResizeService,
HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
    imports: [

        RichTextEditorAllModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor id='defaultRTE' [enableResize]="true">

```

```

        <ng-template #valueTemplate>
            <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content.
                Users can format their content using standard toolbar
commands.</p>
        </ng-template>
    </ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, ResizeService,
HtmlEditorService]
  })
  export class AppComponent {
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Specifying the Minimum and Maximum width and height for Resize

To have a restricted resizable area for the Rich Text Editor, you need to specify the min-width, max-width, min-height, and max-height CSS properties for the control's wrapper element. By default, the control is capable of resizing upto the current viewport. The `e-richtexteditor` CSS class will be available in the component's wrapper and can be used for applying the above mentioned styles.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, ResizeService,
HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    ButtonModule,
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [enableResize]="true">
    <ng-template #valueTemplate>
      <p>The Rich Text Editor component is WYSIWYG ("what you see is
what you get") editor that provides the best user experience to create and
update the content.
          Users can format their content using standard toolbar
commands.</p>
    </ng-template>
  </ejs-richtexteditor>`,

```

```

        providers: [ToolbarService, LinkService, ImageService, ResizeService,
        HtmlEditorService]
    })
    export class AppComponent {
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Number and Bullet Format Lists

This feature allows the user to change the appearance of the Numbered and Bulleted lists. Users can also apply different numbering or bullet formats lists such as lowercase greek, upper Alpha, square and circles. You can also customize the style type of the lists to be populated in the dropdown from the toolbar by using the `numberFormatList` and `bulletFormatList` properties in the Rich Text Editor.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    FormsModule,
    ButtonModule,
    ReactiveFormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools'>
    </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService]
})
export class AppComponent {
  public tools = {items: ['NumberFormatList', 'BulletFormatList']};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Xhtml validation in Angular Rich text editor component

The editor provides an option to validate the source content of the Rich Text Editor against the XHTML standard using the 'enableXhtml' property. When you enter or modify content in the editor, it continuously checks the XHTML source content and removes elements and attributes that are not valid.

The editor checks the following settings on validation:

Attributes

- Must be specified in lowercase.
- Proper use of quotation marks around the attributes.
- Must be valid attributes for corresponding HTML element.
- All the required attributes must be included in the HTML element.

HTML Elements

- Must be in lowercase.
- All opening tags must be closed.
- Allows only the valid HTML elements.
- Elements must be properly nested.
- All elements must have one root element.
- Should not use inline elements inside the block elements.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [enableXhtml] = 'xhtml'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public xhtml = true;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Inline mode in Angular Rich text editor component

This is the inline example for the Rich Text Editor. For this you must set the [inlineMode](#) property.

Inline edition allows you to select any editable element or click the element on the page and edit it in-place.

Inline editing is a true WYSIWYG formation and on the contrary to Rich Text Editor HTML/MD editing, the styles that are used for edited content comes directly from the document stylesheet. This means that inline editors ignore the default Rich Text Editor content styles.

Show on select/click

Enabling the [onSelection](#) option of inlineMode makes the inline Rich Text Editor to appear. You can select the text in the editable area otherwise the inline Rich Text Editor will be appear once click into the editable area.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
/**
 * RTE Inline mode Sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='inlineRTE' #inlineRTE
[inlineMode]='inlineMode' [toolbarSettings]='toolbarSettings'
[format]='format' [fontFamily]='fontFamily'>
  <ng-template #valueTemplate>
    <p>The sample is configured with inline mode of editor.
Initially, the editor is rendered without a <a
href="https://ej2.syncfusion.com/home/" target="_blank">toolbar</a>. The
toolbar becomes visible only when the content is selected.</p>
  </ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
QuickToolbarService]
})
export class AppComponent {
  public inlineMode: object = { enable: true, onSelection: true };
```



```

public toolbarSettings: Object = {
  items: ['Bold', 'Italic', 'Underline',
    'Formats', '-', 'Alignments', 'OrderedList', 'UnorderedList',
    'CreateLink']
};
public format: Object = {
  width: 'auto'
};
public fontFamily: Object = {
  width: 'auto'
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Paste cleanup in Angular Rich text editor component

The Rich Text Editor allows you to reduce the effort while converting the Microsoft Word content to HTML format with format and styles.

MS Word to HTML

By default, Rich Text Editor consider the following processes on paste content from Microsoft Word.

List conversion: The list elements copied from the Microsoft Word document contains paragraph tags with styles and classes. The list elements are converted to standard HTML list elements by referring the styles and class names in the paragraph tags.

Converting style: The styles of the elements copied from the Microsoft Word document are converted to standard CSS styles and added as inline styles for each respective element.

Tags and comments: The Microsoft Word specific XML tags and comments are removed when cleanup on paste.

Paste cleanup

You can control the formatting and styles on pasting the content to the editor using the pasteCleanup settings property. The following settings are available to clean up the content:

API	Description	Default Value	Type
prompt	To invoke prompt dialog with paste options on pasting the content in editor.	false	boolean
plainText	To paste the content as plain text.	false	boolean
keepFormat	To keep the same format with copied content.	true	boolean
deniedTags	To ignore the tags when pasting HTML content.	null	string[]
deniedAttrs	To paste the content by filtering out these attributes from the content.	null	string[]

| [allowedStyleProps](#) | To paste the content by accepting these style attributes and removing other style attributes. | ['background', 'background-color', 'border', 'border-bottom', 'border-left', 'border-radius', 'border-right', 'border-style', 'border-top', 'border-width', 'clear', 'color', 'cursor', 'direction', 'display', 'float', 'font', 'font-family', 'font-size', 'font-weight', 'font-style', 'height', 'left', 'line-height', 'margin', 'margin-top', 'margin-left', 'margin-right', 'margin-bottom', 'max-height', 'max-width', 'min-height', 'min-width', 'overflow', 'overflow-x', 'overflow-y', 'padding', 'padding-bottom', 'padding-left', 'padding-right', 'padding-top', 'position', 'right', 'table-layout', 'text-align', 'text-decoration', 'text-indent', 'top', 'vertical-align', 'visibility', 'white-space', 'width'] | string[] |

Rich Text Editor features are segregated into individual feature-wise modules. To use paste cleanup, inject paste cleanup module using the `RichTextEditor.Inject(PasteCleanup)`.

Prompt dialog

When `prompt` is set to true, pasting the content in the editor will open a dialog box that contains three options `Keep`, `Clean`, and `Plain Text` as radio buttons:

1. `Keep`: Radio button to keep the same format with copied content.
2. `Clean`: Radio button to clear all the style formats with copied content.
3. `Plain Text`: Radio button to paste the copied content as plain text without any formatting or style (including the removal of all tags).

When `prompt` value is set true, the API properties [plainText](#) and [keepFormat](#) will not be considered for processing when pasting the content.

Paste as plain text

When `plainText` is set to true, the copied content will be converted as plain text by removing all the HTML tags and styles applied to it and only the plain text is pasted in the editor.

When `plainText` value is set true, the API property [prompt](#) should be set to false, and [keepFormat](#) will not be considered for processing when pasting the content.

Keep format

When `keepFormat` is set to true, the copied content will maintain all the style formatting allowed in the `allowedStyleProps` on pasting the content in the editor.

When `keepFormat` is set to false, the style in the copied content will be removed without considering the allowed styles in the `allowedStyleProps` when pasting the content in the editor.

When `keepFormat` value is set true, the API property [prompt](#) and [plainText](#) should be set to false.

Denied tags

When `deniedTags` values are set, the tags that matches the 'denied tags' list will be removed on pasting the copied content in the editor. For Example,

1. `'a'`: Paste the content by filtering out anchor tags.
2. `'a[!href]'`: Paste the content by filtering out anchor tags that do not have the 'href' attribute.
3. `'a[href, target]'`: Paste the content by filtering out anchor tags that have the 'href' and 'target' attributes.

Denied attributes

When the `deniedAttrs` values are set, the attributes that matches the 'denied attributes' list will be removed on pasting the copied content in the editor. For Example,

`'id', 'title'`: This will remove the attributes 'id' and 'title' from all tags.

Allowed style properties

By default, the following basic styles are allowed on pasting the content to the editor.

`['background', 'background-color', 'border', 'border-bottom', 'border-left', 'border-radius', 'border-right', 'border-style', 'border-top', 'border-width', 'clear', 'color', 'cursor', 'direction', 'display', 'float', 'font', 'font-family', 'font-size', 'font-weight', 'font-style', 'height', 'left', 'line-height', 'margin', 'margin-top', 'margin-left', 'margin-right', 'margin-bottom', 'max-height', 'max-width', 'min-height', 'min-width', 'overflow', 'overflow-x', 'overflow-y', 'padding', 'padding-bottom', 'padding-left', 'padding-right', 'padding-top', 'position', 'right', 'table-layout', 'text-align', 'text-decoration', 'text-indent', 'top', 'vertical-align', 'visibility', 'white-space', 'width']`

When you configure `allowedStyleProps`, the styles, which matches the 'allowed style properties' list are allowed, all other style properties will be removed on pasting the content in the editor.

For Example,

`allowedStyleProps: ['color', 'margin']`: This will allow only the style properties 'color' and 'margin' in each pasted element.

In the following example, the paste cleanup related settings are explained with its module configuration

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component } from '@angular/core';
import { ToolbarService, HtmlEditorService, PasteCleanupService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' [toolbarSettings]='tools' [pasteCleanupSettings]='pasteCleanupSettings'>
    <ng-template #valueTemplate>
      <p>RichTextEditor is a WYSIWYG editing control which will reduce the effort for users while trying to express their formatting word content as HTML or Markdown format.</p>
      <p><b>Paste Cleanup properties:</b></p>
      <ul>
        <li>
          <p>prompt - specifies whether to enable the prompt when pasting in RichTextEditor.</p>
        </li>
        <li>
```

```

        <p>plainText - specifies whether to paste as plain text or not
in RichTextEditor.</p>
      </li>
      <li>
        <p>keepFormat- specifies whether to keep or remove the format
        when pasting in RichTextEditor.</p>
      </li>
      <li>
        <p>deniedTags - specifies the tags to restrict when pasting in
        RichTextEditor.</p>
      </li>
      <li>
        <p>deniedAttributes - specifies the attributes to restrict when
        pasting in RichTextEditor.</p>
      </li>
      <li>
        <p>allowedStyleProperties - specifies the allowed style
        properties when pasting in RichTextEditor.</p>
      </li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, HtmlEditorService, PasteCleanupService]
})
export class AppComponent {
  public tools: object = {
    type: 'MultiRow',
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public pasteCleanupSettings: object = {
    prompt: true,
    plainText: false,
    keepFormat: false,
    deniedTags: ['a'],
    deniedAttrs: ['class', 'title', 'id'],
    allowedStyleProps: ['color', 'margin', 'font-size']
  };
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Mention integration in Angular Rich text editor component

By integrating the [Mention](#) component with a Rich Text Editor, users can easily mention or tag other users or objects from the suggested list without having to manually type out their names or other identifying information.

The [target](#) property of the Mention component allows you to specify the **ID** of the content editable div element within the Rich Text Editor that you want to bind the Mention component to. This allows you to enable the Mention functionality within the Rich Text Editor, so that users can mention or tag other users or objects from the suggested list while editing the text.

When the user types the **@** symbol followed by a character, the Rich Text Editor will display a list of suggestions for items that the user can select from. The user can then select an item from the list by clicking on it, or by typing the name of the item they want to tag.

In the following sample, configured the following properties with popup dimensions.

- [allowSpaces](#) - Allow to continue search action if user enter space after mention character while searching.
- [suggestionCount](#) - The maximum number of items that will be displayed in the suggestion list.
- [itemTemplate](#) - Used to display the customized appearance in suggestion list.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
/**
 * Rich Text Editor Custom-Toolbar Sample
 */
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, HtmlEditorService, ImageService, LinkService } from '@syncfusion/ej2-angular-richtexteditor';
import { Mention } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    RichTextEditorAllModule,
    MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id="mention_integration"
placeholder="Type @ and tag the name" (actionBegin)="onActionBegin($event)">
  <ng-template #valueTemplate>
    <p>Hello <span contenteditable="false" class="e-mention-chip"><a
href="mailto:maria@gmail.com"
title="maria@gmail.com">&#64;Maria</a></span>&#8203;</p>
    <p>Welcome to the mention integration with rich text editor demo.
Type <code>&#64;</code> character and tag user from the suggestion list. </p>
  </ng-template>
</ejs-richtexteditor>
```

```

<ejs-mention #mention [dataSource]='data' target='#mention_integration_rte-
edit-view' [fields]='fieldsData' [suggestionCount]="8"
[showMentionChar]="false" [allowSpaces]="true"
popupWidth='250px' popupHeight='200px'>
  <ng-template #itemTemplate let-data>
    <table>
      <tr>
        <td>
          <div id="mention-TemplateList">
            
            <span class="e-badge e-badge-success e-badge-overlap e-
badge-dot e-badge-bottom {{data.Status}}"></span>
          </div>
        </td>
        <td class="mentionNameList">
          <span class="person">{{data.Name}}</span>
          <span class="email">{{data.EmailId}}</span>
        </td>
      </tr>
    </table>
  </ng-template>
  <ng-template #displayTemplate let-data>
    <a href="mailto:{{data.EmailId}}"
title="{{data.EmailId}}">&#64;{{data.Name}}</a>
  </ng-template>
</ejs-mention>
`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
} )
export class AppComponent {
  @ViewChild('mention') mentionObj: Mention;
  public toolbarSettings: object = {
    items: ['Bold', 'Italic', 'Underline', '|', 'Formats', 'Alignments',
'OrderedList',
'UnorderedList', '|', 'CreateLink', 'Image', '|', 'SourceCode', '|',
'Undo', 'Redo'
]
};
  public data: { [key: string]: Object }[] = [
    { Name: "Selma Rose", Status: "active", Eimg: "2", EmailId:
"selma@gmail.com" },
    { Name: "Maria", Status: "active", Eimg: "1", EmailId:
"maria@gmail.com" },
    { Name: "Russo Kay", Status: "busy", Eimg: "8", EmailId:
"russo@gmail.com" },
    { Name: "Camden Kate", Status: "active", Eimg: "9", EmailId:
"camden@gmail.com" },
    { Name: "Robert", Status: "busy", Eimg: "dp", EmailId:
"robert@gmail.com" },
    { Name: "Garth", Status: "active", Eimg: "7", EmailId:
"garth@gmail.com" },
    { Name: "Andrew James", Status: "away", Eimg: "pic04", EmailId:
"noah@gmail.com" },
    { Name: "Olivia", Status: "busy", Eimg: "5", EmailId:
"olivia@gmail.com" },

```

```

    { Name: "Sophia", Status: "away", Eimg: "6", EmailId:
"sophia@gmail.com" },
    { Name: "Margaret", Status: "active", Eimg: "3", EmailId:
"margaret@gmail.com" },
    { Name: "Ursula Ann", Status: "active", Eimg: "dp", EmailId:
"ursula@gmail.com" },
    { Name: "Laura Grace", Status: "away", Eimg: "4", EmailId:
"laura@gmail.com" },
    { Name: "Albert", Status: "active", Eimg: "pic03", EmailId:
"albert@gmail.com" },
    { Name: "William", Status: "away", Eimg: "10", EmailId:
"william@gmail.com" }
  ];
  public fieldsData: { [key: string]: string } = { text: 'Name' };
  onActionBegin(args: any) {
    if (args.requestType === 'EnterAction' &&
this.mentionObj.element.classList.contains('e-popup-open')) {
      args.cancel = true;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

[View Sample](#)

See Also

- [Mention](#)

Enter key in Angular Rich text editor component

Rich Text Editor allows to customize the tag that is inserted when pressing the enter key and shift + enter key in the Rich Text Editor.

Enter key customization

By default, the `<p>` tag will be created while pressing the enter key. The enter key can be customized by using the [enterKey](#) property, where the possible tags that can be used to customize are `<p>`, `<div>`, and `
`.

When the enter key is customized with any of the possible values, pressing the enter key in the editor will create a new tag that is configured. Also, when the enter key is configured the default value of the Rich Text Editor will also change respectively with the configured values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'

```

```

import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { createElement } from '@syncfusion/ej2-base';
import { RichTextEditorComponent, ToolbarService, LinkService,
ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-
richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from
 '@syncfusion/ej2-angular-dropdowns';
@Component({
imports: [

    RichTextEditorAllModule,
    DialogModule,
    DropDownListModule
],
standalone: true,
selector: 'app-root',
template: `<div class="control-section">
    <table class='api'>
        <tbody>
            <tr>
                <td>
                    <div>
                        <ejs-dropdownlist id='enterOption' #enterOption
                        [dataSource]='enterOptionData'
(change)='enterChange()'
                        [value]='enterValue' [fields]='fields'
[popupHeight]='height'
                        [placeholder]='enterPlaceHolder'
[floatLabelType]='floatLabel'></ejs-dropdownlist>
                    </div>
                </td>
            </tr>
        </tbody>
    </table>
    <br>
    <ejs-richtexteditor id='defaultRTE' #defaultRTE [height]='rteHeight'>
        <ng-template #valueTemplate>
            <p>In Rich text Editor, the enter key and shift + enter key
actions can be customized using the enterKey and shiftEnterKey APIs. And the
possible values are as follows:</p><ul><li>P - When 'P' is configured,
pressing enter or shift + enter will create a 'p' tag</li><li>DIV - When
'DIV' is configured, pressing enter or shift + enter will create a 'div'
tag</li><li>BR - When 'BR' is configured, pressing enter or shift + enter
will create a 'br' tag</li></ul>
        </ng-template>
    </ejs-richtexteditor>
</div>`,
encapsulation: ViewEncapsulation.None,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
    @ViewChild('defaultRTE')
    public rteObj?: RichTextEditorComponent;
    @ViewChild('enterOption')
    public enterObj?: DropDownListComponent;

```



```

public enterOptionData: { [key: string]: Object }[] = [
  { Text: 'Create a new <p>', Value: 'P' },
  { Text: 'Create a new <div>', Value: 'DIV' },
  { Text: 'Create a new <br>', Value: 'BR' }
];
public enterPlaceholder: string = 'When pressing the enter key';
public floatLabel: string = 'Always';
public fields: FieldSettingsModel = { text: 'Text', value: 'Value' };
public rteHeight = 220;
public height: string = '200px';
public enterValue: string = 'P';
public enterChange(): void {
  if (this.enterObj!.value === 'P') {
    this.rteObj!.enterKey = 'P';
    this.rteObj!.value = `<p>In Rich text Editor, the enter key
and shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:</p><ul><li>P -
When 'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
  } else if (this.enterObj!.value === 'DIV') {
    this.rteObj!.enterKey = 'DIV';
    this.rteObj!.value = `<div>In Rich text Editor, the enter key
and shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:</div><ul><li>P -
When 'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
  } else if (this.enterObj!.value === 'BR') {
    this.rteObj!.enterKey = 'BR';
    this.rteObj!.value = `In Rich text Editor, the enter key and
shift + enter key actions can be customized using the enterKey and
shiftEnterKey APIs. And the possible values are as follows:<ul><li>P - When
'P' is configured, pressing enter or shift + enter will create a 'p'
tag</li><li>DIV - When 'DIV' is configured, pressing enter or shift + enter
will create a 'div' tag</li><li>BR - When 'BR' is configured, pressing enter
or shift + enter will create a 'br' tag</li></ul>`;
  }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Shift-Enter key customization

By default, the `
` tag will be created while pressing the shift + enter key. The shift + enter key can be customized by using the [shiftEnterKey](#) property where the possible tags that can be used to customize are `
`, `<p>`, `<div>`.

When the shift + enter key is customized with any of the possible values, pressing the shift + enter key in the editor will create a new tag that is configured. Also, when the shift + enter key is configured the default value of the Rich Text Editor will change respectively with the configured values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { createElement } from '@syncfusion/ej2-base';
import { RichTextEditorComponent, ToolbarService, LinkService,
ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
import { DropDownListComponent, FieldSettingsModel } from
 '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule,
    DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <table class='api'>
      <tbody>
        <tr>
          <td>
            <div>
              <ejs-dropdownlist id='shiftEnterOption'
#shiftEnterOption
                                [dataSource]='shiftEnterData'
              (change)='shiftEnterChange()'
                                [value]='shiftEnterValue' [fields]='fields'
              [popupHeight]='height'
                                [placeholder]='shiftEnterPlaceHolder'
              [floatLabelType]='floatLabel'></ejs-dropdownlist>
            </div>
          </td>
        </tr>
      </tbody>
    </table>
    <br>
    <ejs-richtexteditor id='defaultRTE' #defaultRTE [height]='rteHeight'>
      <ng-template #valueTemplate>
        <p>In Rich text Editor, the enter key and shift + enter key
actions can be customized using the enterKey and shiftEnterKey APIs. And the
possible values are as follows:</p><ul><li>P - When 'P' is configured,
pressing enter or shift + enter will create a 'p' tag</li><li>DIV - When
'DIV' is configured, pressing enter or shift + enter will create a 'div'
tag</li><li>BR - When 'BR' is configured, pressing enter or shift + enter
will create a 'br' tag</li></ul>
      </ng-template>
    </ejs-richtexteditor>
  `
})
export class AppRootComponent {
}
```

```

        </ng-template>
    </ejs-richtexteditor>
</div>`,
encapsulation: ViewEncapsulation.None,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
    @ViewChild('defaultRTE')
    public rteObj?: RichTextEditorComponent;
    @ViewChild('shiftEnterOption')
    public shiftEnterObj?: DropDownListComponent;
    public shiftEnterData: { [key: string]: Object }[] = [
        { Text: 'Create a new <br>', Value: 'BR' },
        { Text: 'Create a new <div>', Value: 'DIV' },
        { Text: 'Create a new <p>', Value: 'P' }
    ];
    public shiftEnterPlaceholder: string = 'When pressing the shift +
enter key';
    public floatLabel: string = 'Always';
    public fields: FieldSettingsModel = { text: 'Text', value: 'Value' };
    public rteHeight = 220;
    public height: string = '200px';
    public shiftEnterValue: string = 'BR';
    public shiftEnterChange(): void {
        if (this.shiftEnterObj!.value === 'BR') {
            this.rteObj!.shiftEnterKey = 'BR';
        } else if (this.shiftEnterObj!.value === 'DIV') {
            this.rteObj!.shiftEnterKey = 'DIV';
        } else if (this.shiftEnterObj!.value === 'P') {
            this.rteObj!.shiftEnterKey = 'P';
        }
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Iframe in Angular Rich text editor component

When the [iframeSettings](#) option is enabled, the Rich Text Editor creates the iframe element as the content area on control initialization. It is used to display and editing the content in content area. The editor will display only the body tag of a <iframe> document.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
/**
 * RTE IFrame Sample

```

```

    */
    import { Component } from '@angular/core';
    import { ToolbarService, LinkService, ImageService, HtmlEditorService }
    from '@syncfusion/ej2-angular-richtexteditor';
    @Component({
    imports: [

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
        selector: 'app-root',
        template: `<ejs-richtexteditor id='iframeRTE'
[iframeSettings]='iframe'></ejs-richtexteditor>`,
        providers: [ToolbarService, LinkService, ImageService,
    HtmlEditorService]
    })
    export class AppComponent {
        public iframe: object = { enable: true };
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

IFrame attributes

The editor allows you to pass an additional attribute to body tag of a <iframe> element using attributes fields of the [iframeSettings](#) property. This property contains name/value pairs in string format. It is used to override the default appearance of the content area.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
    /**
     * RTE IFrame attributes Sample
     */
    import { Component } from '@angular/core';
    import { ToolbarService, LinkService, ImageService, HtmlEditorService } from
    '@syncfusion/ej2-angular-richtexteditor';
    @Component({
    imports: [

        RichTextEditorAllModule,
        DialogModule
    ],
    standalone: true,
        selector: 'app-root',

```

```

    template: `<ejs-richtexteditor id='iframeRTE'
[iframeSettings]='iframe'></ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
  })
  export class AppComponent {
    public iframe: object = {
      enable: true,
      attributes: {
        readonly: 'readonly'
      }
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Adding external CSS/Script file

The editor offers you to add external CSS file to style the < iframe > element. Easily change the appearance of editor's content using an external CSS file using [styles](#) field in the iframeSettings property.

Likewise, add the external script file to the < iframe > element using the [scripts](#) field of iframeSettings to provide the additional functionalities to the RichTextEditor.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
/**
 * RTE IFrame Sample
 */
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService }
from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='iframeRTE'
[iframeSettings]='iframe'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  public iframe: object = {
    enable: true,

```

```

        attributes: {
            readonly: 'readonly'
        },
        resources: {
            scripts: [''], // script.js
            styles: [''] // css
        }
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can also explore our [iframe in Angular Rich Text Editor exampleLink to the Video](#) that shows how to render the iframe in Angular Rich Text Editor.

Third party integration in Angular Rich text editor component

The Rich Text Editor can be integrated with third-party to suite the application scenario.

To get start quickly with Third-Party Integration for Angular Rich Text Editor component, refer to the video below.

CodeMirror Integration

RichTextEditor comes with a basic HTML source editor through the view-source property. CodeMirror plugin can be used to highlight the syntax of HTML. CodeMirror plugin for Rich Text Editor makes editing of HTML source code with a pleasant experience.

Import necessary CSS and JS files of CodeMirror to the HTML page.

Required JS files of code mirror.

`typescript

```
<script src="scripts/CodeMirror/codemirror.js" type="text/javascript"></script>
```

```
<script src="scripts/CodeMirror/javascript.js" type="text/javascript"></script>
```

```
<script src="scripts/CodeMirror/css.js" type="text/javascript"></script>
```

```
<script src="scripts/CodeMirror/htmlmixed.js" type="text/javascript"></script>
```

,

Required CSS file of code mirror

`typescript

```
<link href="scripts/CodeMirror/codemirror.min.css" rel="stylesheet" />
```

,

Add a custom icon for HTML source editor in the toolbar of Rich Text Editor using the template option of ToolbarSettings, define the code mirror plugins, and then pass the Rich Text Editor content as argument in the [actionComplete](#) event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, RichTextEditorComponent, CountService } from '@syncfusion/ej2-angular-richtexteditor';
import { createElement, addClass, removeClass, Browser } from '@syncfusion/ej2-base';
@Component({
  imports: [
    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #toolsRTE id='alltoolRTE'
[toolbarSettings]='tools' [showCharCount]='true'
(actionComplete)='actionCompleteHandler($event)' [maxLength]='maxLength'>
  <ng-template #valueTemplate>
    <p>The Rich Text Editor is WYSIWYG ("what you see is what you get")
editor useful to create and edit content, and return the valid <a
href="https://ej2.syncfusion.com/home/" target="_blank">HTML markup</a> or <a
href="https://ej2.syncfusion.com/home/" target="_blank">markdown</a> of the
content</p>
    <p><b>Toolbar</b></p>
    <ol>
      <li>
        <p>Toolbar contains commands to align the text, insert
link, insert image, insert list, undo/redo operations, HTML view, etc </p>
      </li>
      <li>
        <p>Toolbar is fully customizable </p>
      </li>
    </ol>
    <p><b>Links</b></p>
    <ol>
      <li>
        <p>You can insert a hyperlink with its corresponding
dialog </p>
      </li>
      <li>
        <p>Attach a hyperlink to the displayed text. </p>
      </li>
      <li>
        <p>Customize the quick toolbar based on the hyperlink
</p>
      </li>
    </ol>
    <p><b>Image.</b></p>
    <ol>
      <li>

```

```

        <p> Allows you to insert images from an online source as
well as the
        local computer </p>
    </li>
    <li>
        <p> You can upload an image </p>
    </li>
    <li>
        <p>Provides an option to customize quick toolbar for an
image </p>
    </li>
</ol>
</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService,
CountService]
}))
export class AppComponent {
@ViewChild('toolsRTE') public rteObj?: RichTextEditorComponent;
public tools: object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
};
public maxLength: number = 1000;
public textArea?: HTMLElement;
public myCodeMirror?: any;
ngAfterViewInit(): void {
let rteObj: RichTextEditorComponent = this.rteObj as any;
setTimeout(() => { this.textArea = (rteObj.contentModule as
any).getEditPanel() as HTMLElement; }, 600);
}
public mirrorConversion(e?: any): void {
let id: string = this.rteObj!.getID() + 'mirror-view';
let mirrorView: HTMLElement = this.rteObj!.element.querySelector('#' + id) as
HTMLElement;
let charCount: HTMLElement = this.rteObj!.element.querySelector('.e-rte-
character-count') as HTMLElement;
if (e.targetItem === 'Preview') {
    this.textArea!.style.display = 'block';
    mirrorView.style.display = 'none';
    this.textArea!.innerHTML = this.myCodeMirror.getValue();
    charCount.style.display = 'block';
} else {
    if (!mirrorView) {
        mirrorView = createElement('div', { className: 'e-content' });
        mirrorView.id = id;
        this.textArea!.parentNode!.appendChild(mirrorView);
    } else {
        mirrorView.innerHTML = '';
    }
    this.textArea!.style.display = 'none';
    mirrorView.style.display = 'block';
}

```



```

        this.renderCodeMirror(mirrorView, this.rteObj!.value);
        charCount.style.display = 'none';
    }
}

public renderCodeMirror(mirrorView: HTMLElement, content: string): void {
    // this.myCodeMirror = CodeMirror(mirrorView, {
    //     value: content,
    //     lineNumbers: true,
    //     mode: 'text/html',
    //     lineWrapping: true,
    // });
}

public actionCompleteHandler(e: any): void {
    if (e.targetItem && (e.targetItem === 'SourceCode' || e.targetItem ===
    'Preview')) {
        (this.rteObj!.sourceCodeModule.getPanel() as
    HTMLTextAreaElement).style.display = 'none';
        this.mirrorConversion(e);
    } else {
        setTimeout(() => { this.rteObj!.toolbarModule.refreshToolbarOverflow();
    }, 400);
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Integrate `embedly`

This can be achieved by binding the [actionComplete](#) event to the toolbar items in the [toolbarSettings](#) property. In the event handler, create an element and add the appropriate class. The below script is have to add in the sample to embed the content,

Include **embedly** javascript.

`typescript

```
<script src="https://cdn.embedly.com/widgets/platform.js" charset="UTF-8"></script>
```

,

The above script is added to the page.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({

```

```

imports: [
    RichTextEditorAllModule
],
standalone: true,
selector: 'app-root',
template: `<ejs-richtexteditor id='defaultRTE' #sample
[toolbarSettings]='toolbarSettings'
(actionComplete)='actionComplete($event)'>
    <ng-template #valueTemplate>
        <p>The Rich Text Editor triggers events based on its actions. </p>
        <p>The events can be used as an extension point to perform custom
operations.</p>
        <ul>
            <li>created - Triggers when the component is rendered.</li>
            <li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
            <li>focus - Triggers when RTE is focused in.</li>
            <li>blur - Triggers when RTE is focused out.</li>
            <li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
            <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
            <li>destroyed - Triggers when the component is destroyed.</li>
        </ul>
    </ng-template></ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService
]
})
export class AppComponent {
    @ViewChild('sample') public rteObj?: RichTextEditorComponent;
    public toolbarSettings: Object = {
        items: ['createLink'];
    };
    public actionComplete(args: any): void {
        if (<String>args.requestType === 'Links') {
            if (args.elements[0].parentNode &&
(<Element>args.elements[0].parentNode).tagName === 'A'){
                const emberEle: HTMLElement = document.createElement('blockquote');
                emberEle.setAttribute('class', 'embedly-card');
                emberEle.appendChild(args.elements[0].parentElement);
                emberEle.appendChild(document.createElement('p'));
                args.range.insertNode(emberEle);
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Exec command in Angular Rich text editor component

The execCommand is used to perform command for the modification of content in editable area. The list of valid execCommand are given in the below table.

Commands	Description	Code snippets
-----	-----	-----
bold	Bold the selected content in the Rich Text Editor.	<code> rteObj.executeCommand('bold'); </code>
italic	The selected text will be italics.	<code> rteObj.executeCommand('italic'); </code>
underline	Underline the selected text in the Rich Text Editor.	<code> rteObj.executeCommand('underline'); </code>
strikeThrough	Apply single line strike through formatting for the selected text.	<code> rteObj.executeCommand('strikeThrough'); </code>
superscript	Makes the selected text as superscript (higher).	<code> rteObj.executeCommand('superscript'); </code>
subscript	Makes the selected text as subscript (lower).	<code> rteObj.executeCommand('subscript'); </code>
uppercase	Change the case of selected text to upper in the content.	<code> rteObj.executeCommand('uppercase'); </code>
lowercase	Change the case of selected text to lower in the content.	<code> rteObj.executeCommand('lowercase'); </code>
fontColor	Apply the specified font color for the selected text.	<code> rteObj.executeCommand('fontColor', 'yellow'); </code>
fontName	Apply the specified font name for the selected text.	<code> rteObj.executeCommand('fontName', 'Arial'); </code>
fontSize	Apply the specified font size for the selected text.	<code> rteObj.executeCommand('fontSize', '10pt'); </code>
formatBlock	Apply the specified format styles for the selected text.	<code> rteObj.executeCommand('formatBlock', 'H1'); </code>
backColor	Apply the specified background color the selected text.	<code> rteObj.executeCommand('backColor', 'red'); </code>
justifyCenter	Align the content with center margin.	<code> rteObj.executeCommand('justifyCenter'); </code>
justifyFull	Align the content with justify margin.	<code> rteObj.executeCommand('justifyFull'); </code>
justifyLeft	Align the content with left margin.	<code> rteObj.executeCommand('justifyLeft'); </code>
justifyRight	Align the content with right margin.	<code> rteObj.executeCommand('justifyRight'); </code>
undo	Allows to undo the actions.	<code> rteObj.executeCommand('undo'); </code>
createLink	Creates a hyperlink to a text or image to a specific location in the content.	<code> rteObj.executeCommand('createLink',{ text: 'Links', url: 'http://', title: 'Link' }); </code>
indent	Allows to increase the indent level of the content.	<code> rteObj.executeCommand('indent'); </code>

| insertHTML | Insert the html content to the current cursor position. |
`rteObj.executeCommand('insertHTML', 'inserted an html');` |

| insertOrderedList | Create a new list item(numbered). |
`rteObj.executeCommand('insertOrderedList');` |

| insertUnorderedList | Create a new list item(bulleted). |
`rteObj.executeCommand('insertUnorderedList');` |

| outdent | Allows to decrease the indent level of the content. |
`rteObj.executeCommand('outdent');` |

| redo | Allows to redo the actions | `rteObj.executeCommand('redo');` |

| removeFormat | remove all formatting styles (such as bold, italic, underline, color, superscript, subscript, and more) from currently selected text. | `rteObj.executeCommand('removeFormat');` |

| insertText | Insert text to the current cursor position. | `rteObj.executeCommand('insertText', 'inserted a text');` |

| insertImage | Insert an image to the current cursor position. |
`rteObj.executeCommand('insertImage', { url: 'https://ej2.syncfusion.com/javascript/demos/src/rich-text-editor/images/RTEImage-Feather.png', cssClass: 'rte-img' });` |

| copyFormatPainter | Copy the format of selected text and apply it to another text. |
`rteObj.executeCommand('copyFormatPainter', formatPainterSettings);` |

| applyFormatPainter | Apply the copied format to the selected text. |
`rteObj.executeCommand('applyFormatPainter');` |

| escapeFormatPainter | Remove the previously copied format and disable the sticky mode |
`rteObj.executeCommand('escapeFormatPainter');` |

Note: The 'ExecuteCommand' public method is not supported in Syncfusion Markdown Editor

Style in Angular Rich text 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 Rich Text Editor's content

Use the following CSS to customize the default Rich Text Editor's content properties like font-family, font-size and color.

`CSS

/ To change font family and font size /

```
.e-richtexteditor .e-rte-content .e-content,
.e-richtexteditor .e-source-content .e-content {
font-size: 20px;
font-family: Segoe ui;
}
```

/ To change font color and content background /

```
.e-richtexteditor .e-rte-content,
.e-richtexteditor .e-source-content {
background: seashell;
color: blue;
}
`
```

Customizing the Rich Text Editor's toolbar

Use the following CSS to customize the default color in the Rich Text Editor's toolbar icon.

`CSS

/ To change font color for toolbar icon /

```
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-icons,
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-icons:active {
color: red;
}
`
```

/ To change font color for toolbar button /

```
.e-toolbar .e-tbar-btn,
.e-toolbar .e-tbar-btn:active,
.e-toolbar .e-tbar-btn:hover {
color: red;
}
`
```

/ To change font color for toolbar button in active state/

```
.e-richtexteditor .e-rte-toolbar .e-toolbar-item .e-dropdown-btn.e-active .e-icons, .e-richtexteditor .e-
rte-toolbar .e-toolbar-item .e-dropdown-btn.e-active .e-rte-dropdown-btn-text {
color: red;
}
`
```

/ To change font color for expanded toolbar items /

```
.e-richtexteditor .e-rte-toolbar .e-toolbar-extended .e-toolbar-item .e-tbar-btn .e-icons,
.e-toolbar.e-extended-toolbar .e-toolbar-extended .e-toolbar-item .e-tbar-btn {
color: red;
}
`
```

Customizing the Rich Text Editor's character count

Use the following CSS to customize the default color in the Rich Text Editor's character count.

```
`CSS
/ To change font color, font family, font size and opacity /
.e-richtexteditor .e-rte-character-count {
color: red;
font-family: segoe ui;
font-size: 18px;
opacity: 0.54;
padding-bottom: 2px;
padding-right: 14px;
}
`
```

Keyboard support in Angular Rich text editor component

The Rich Text Editor has full keyboard accessibility that includes shortcuts to open and other actions with toolbar items, drop-down lists and dialogs.

HTML formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with HTML edit mode.

| Actions | Keyboard shortcuts |

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

| **Toolbar focus** | **Alt + f10** |

| **Insert link** | **Ctrl + k** |

| **Insert image** | **Ctrl + Shift + i** |

| **Insert audio** | **Ctrl + Shift + a** |

| **Insert video** | **Ctrl + Alt + v** |

| **Insert table** | **Ctrl + Shift + e** |

| **Undo** | **Ctrl + z** |

| **Redo** | **Ctrl + y** |

| **Copy** | **Ctrl + c** |

| **Cut** | **Ctrl + x** |

| **Paste** | **Ctrl + v** |

| **Bold** | **Ctrl + b** |

| **Italic** | **Ctrl + i** |

| **Underline** | **Ctrl + u** |

| **Strikethrough** | **Ctrl + Shift + s** |

Uppercase	Ctrl + Shift + u
Lowercase	Ctrl + Shift + l
Superscript	Ctrl + Shift + =
Subscript	Ctrl + =
Indents	Ctrl +]
Outdents	Ctrl + [
HTML source	Ctrl + Shift + h
Fullscreen	Ctrl + Shift + f
Exit Fullscreen	Esc
Justify center	Ctrl + e
Justify full	Ctrl + j
Justify left	Ctrl + l
Justify right	Ctrl + r
Clear format	Ctrl + Shift + r
Ordered list	Ctrl + Shift + o
Unordered list	Ctrl + Alt + o
Format Painter Copy	Alt + Shift + c
Format Painter Paste	Alt + Shift + v
Format Painter Escape	Esc

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
  import { Component, ViewChild } from '@angular/core';
  import { RichTextEditorComponent, ToolbarService, HtmlEditorService,
ImageService, QuickToolbarService, LinkService, FormatPainterService } from
'@syncfusion/ej2-angular-richtexteditor';
  @Component({
imports: [

      RichTextEditorAllModule,
      DialogModule
    ],
standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #defaultRTE id='defaultRTE'
[toolbarSettings]='tools' (create)='onCreate($event)'></ejs-richtexteditor>`,

```

```

    providers: [ToolbarService, HtmlEditorService, ImageService,
QuickToolbarService, LinkService, FormatPainterService ]
  })
  export class AppComponent {
    @ViewChild('defaultRTE') rteObj: RichTextEditorComponent | undefined;
    public tools = {
      items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'FormatPainter', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
    };
    onCreate(e: any) {
      document.onkeyup = function (e) {
        if (e.altKey && e.keyCode === 84 /* t */) {
          // press alt+t to focus the component.
          (this as any).rteObj.focusIn();
        }
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Markdown formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with Markdown edit mode

| Actions | Keyboard shortcuts |

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

| Toolbar focus | Alt + f10 |

| Insert link | Ctrl + k |

| Insert image | Ctrl + Shift + i |

| Insert table | Ctrl + Shift + e |

| Undo | Ctrl + z |

| Redo | Ctrl + y |

| Copy | Ctrl + c |

| Cut | Ctrl + x |

| Paste | Ctrl + v |

| Bold | Ctrl + b |

| Italic| Ctrl + i |

| Strikethrough| Ctrl + Shift + s |

| Uppercase| Ctrl + Shift + u |

| Lowercase| Ctrl + Shift + l |

| Superscript| Ctrl + Shift + = |

| Subscript| Ctrl + = |

| Fullscreen| Ctrl + Shift + f |

| Ordered list| Ctrl + Shift + o |

| Unordered list| Ctrl + Alt + o |

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService,
MarkdownEditorService, RichTextEditorComponent } from '@syncfusion/ej2-
angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #defaultRTE id='defaultRTE'
[toolbarSettings]='tools'
[editorMode]='mode' (create)='onCreate($event)'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService,
MarkdownEditorService]
})
export class AppComponent {
  @ViewChild('defaultRTE') rteObj: RichTextEditorComponent | undefined;
  public tools = {
    items: ['Bold', 'Italic', 'StrikeThrough', '|',
      'Formats', 'OrderedList', 'UnorderedList', '|',
      'CreateLink', 'Image', '|', 'Undo', 'Redo']
  };
  public mode = 'Markdown';
  onCreate(e: any) {
    document.onkeyup = function (e) {
      if (e.altKey && e.keyCode === 84 /* t */) {
        // press alt+t to focus the component.
        (this as any).rteObj.focusIn();
      }
    }
  }
}

```

```

    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Custom key config

You can able to customize the key config for the keyboard interaction of Rich Text Editor, using [keyConfig](#) property.

In the below sample, you have customize the bold, italic, underline toolbar action with ctrl+alt+b, ctrl+alt+i and ctrl+alt+u respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule,
    DialogModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor #defaultRTE id='defaultRTE'
[toolbarSettings]='tools' [keyConfig]='keyConfig'
(create)='onCreate($event)'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
})
export class AppComponent {
  @ViewChild('defaultRTE') rteObj: RichTextEditorComponent | undefined;
  public tools = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public keyConfig = {
    'copy': 'ctrl+1',
    'cut': 'ctrl+2',
    'paste': 'ctrl+3'
  }
}

```

```

    }
    onCreate(e: any) {
      document.onkeyup = function (e) {
        if (e.altKey && e.keyCode === 84 /* t */) {
          // press alt+t to focus the component.
          (this as any).rteObj.focusIn();
        }
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

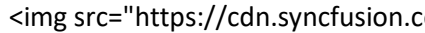
Accessibility in Angular Rich text editor component

The Rich Text Editor 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 Rich Text Editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  alt="Yes" > |

| [Section 508 Support](#) |  alt="Yes" > |

| [Screen Reader Support](#) |  alt="Yes" > |

| [Right-To-Left Support](#) |  alt="Yes" > |

| [Color Contrast](#) |  alt="Intermediate" > |

| [Mobile Device Support](#) |  alt="Yes" > |

| [Keyboard Navigation Support](#) |  alt="Yes" > |

| [Accessibility Checker Validation](#) |  alt="Intermediate" > |

| [Axe-core Accessibility Validation](#) |  alt="Yes" > |

```

<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 toolbar of Rich Text Editor, assigned the role of 'Toolbar' and has the following list of aria attribute.

| Property | Functionalities |

| --- | --- |

| role="toolbar" | This attribute added to the ToolBar element describes the actual role of the element. |

| aria-orientation | Indicates the ToolBar orientation. Default value is **horizontal**. |

| aria-haspopup | Indicates the popup mode of the Toolbar. Default value is false. When popup mode is enabled, attribute value has to be changed to **true**. | |

| aria-disabled | Indicates the disabled state of the ToolBar. |

| aria-owns | Identifies an element to define a visual, functional, or contextual parent/child relationship between DOM elements when the DOM hierarchy cannot represent the relationship. In the Rich Text Editor, the attribute contains the ID of the Rich Text Editor to indicate the popup as a child element. |

For further details of Toolbar ARIA attributes, please look in to [Accessibility of Toolbar](#) documentation.

- The Rich Text Editor element is assigned the role of **application**.

| Property | Functionalities |

| --- | --- |

| role="application" | This attribute added to the Rich Text Editor element describes the actual role of the element. |

| aria-disabled | Indicates the disabled state of the ToolBar. |

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
    import { Component } from '@angular/core';
    import { ToolbarService, LinkService, ImageService, HtmlEditorService }
from '@syncfusion/ej2-angular-richtexteditor';
    @Component({
imports: [

        RichTextEditorAllModule,
        DialogModule

    ],
standalone: true,
    selector: 'app-root',
    template: `<ejs-richtexteditor #defaultRTE id='defaultRTE'
[toolbarSettings]='tools'></ejs-richtexteditor>`,
    providers: [ToolbarService, LinkService, ImageService, HtmlEditorService]
    })
    export class AppComponent {
        public tools = {
            items: ['Bold', 'Italic', 'Underline', 'StrikeThrough',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
        };
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Keyboard interaction

The Rich Text 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 Rich Text Editor component.

HTML formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with HTML edit mode.

| Actions | Keyboard shortcuts |

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

| Toolbar focus | Alt + f10 |

| Insert link | Ctrl + k |

| Insert image | Ctrl + Shift + i |

Insert table	Ctrl + Shift + e
Undo	Ctrl + z
Redo	Ctrl + y
Copy	Ctrl + c
Cut	Ctrl + x
Paste	Ctrl + v
Bold	Ctrl + b
Italic	Ctrl + i
Underline	Ctrl + u
Strikethrough	Ctrl + Shift + s
Uppercase	Ctrl + Shift + u
Lowercase	Ctrl + Shift + l
Superscript	Ctrl + Shift + =
Subscript	Ctrl + =
Indents	Ctrl +]
Outdents	Ctrl + [
HTML source	Ctrl + Shift + h
Fullscreen	Ctrl + Shift + f
Exit Fullscreen	Esc
Justify center	Ctrl + e
Justify full	Ctrl + j
Justify left	Ctrl + l
Justify right	Ctrl + r
Clear format	Ctrl + Shift + r
Ordered list	Ctrl + Shift + o
Unordered list	Ctrl + Alt + o
Format Painter Copy	Alt + Shift + c
Format Painter Paste	Alt + Shift + v
Format Painter Escape	Esc

Markdown formation shortcut key

You can use the following key shortcuts when the Rich Text Editor renders with Markdown edit mode

| Actions | Keyboard shortcuts |

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

| Toolbar focus | Alt + f10 |

| Insert link | Ctrl + k |

| Insert image | Ctrl + Shift + i |

| Insert table | Ctrl + Shift + e |

| Undo | Ctrl + z |

| Redo | Ctrl + y |

| Copy | Ctrl + c |

| Cut | Ctrl + x |

| Paste | Ctrl + v |

| Bold | Ctrl + b |

| Italic | Ctrl + i |

| Strikethrough | Ctrl + Shift + s |

| Uppercase | Ctrl + Shift + u |

| Lowercase | Ctrl + Shift + l |

| Superscript | Ctrl + Shift + = |

| Subscript | Ctrl + = |

| Fullscreen | Ctrl + Shift + f |

| Ordered list | Ctrl + Shift + o |

| Unordered list | Ctrl + Alt + o |

Ensuring accessibility

The Rich Text Editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Rich Text Editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Rich Text Editor component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Add google fonts in Angular Rich text editor component

To use web fonts in RTE, it is not needed for the web fonts to be present in local machine. To add the web fonts to RTE, you need to refer the web font links and add the font names in the [fontFamily](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
imports: [

    RichTextEditorAllModule
],
standalone: true,
selector: 'app-root',
template: `<ejs-richtexteditor id='defaultRTE' #sample
[fontFamily]='fontFamily' [toolbarSettings]='toolbarSettings'>
<ng-template #valueTemplate>
    <p>The Rich Text Editor triggers events based on its actions. </p>
    <p> The events can be used as an extension point to perform custom
operations.</p>
    <ul>
        <li>created - Triggers when the component is rendered.</li>
        <li>change - Triggers only when RTE is blurred and changes are done to
the content.</li>
        <li>focus - Triggers when RTE is focused in.</li>
        <li>blur - Triggers when RTE is focused out.</li>
        <li>actionBegin - Triggers before command execution using toolbar items
or executeCommand method.</li>
        <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
        <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
</ng-template>
</ejs-richtexteditor>`,
providers: [ToolbarService, LinkService, ImageService, HtmlEditorService
]
})
export class AppComponent {
@ViewChild('sample') public rteObj?: RichTextEditorComponent;
public fontFamily: Object = {
    items: [
        {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui", command:
"Font", subCommand: "FontName"},
        {text: "Roboto", value: "Roboto", command: "Font", subCommand:
"FontName"}, // here font is added
        {text: "Great vibes", value: "Great Vibes,cursive", command: "Font",
subCommand: "FontName"}, // here font is added
        {text: "Noto Sans", value: "Noto Sans", command: "Font", subCommand:
"FontName"},
        {text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-impact",
command: "Font", subCommand: "FontName"},
        {text: "Tahoma", value: "Tahoma, Geneva, sans-serif", class: "e-tahoma",
command: "Font", subCommand: "FontName"},
    ]
};

```



```
public toolbarSettings: Object = {
  items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
    'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
    'LowerCase', 'UpperCase', '|',
    'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
    'Outdent', 'Indent', '|',
    'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
    'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The below font style links are referred in the page.

`typescript

<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Roboto">

<link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Great+Vibes">

,

In the above sample, you can see that you have added two Google web fonts (**Roboto** and **Great vibes**) to RTE.

Change default font family in Angular Rich text editor component

By using [default](#) property, you can change the default font-family of the RTE. To change the font-family of the RTE content while loading, you need to give the font-family in the style section with the help of [cssClass](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample
[fontFamily]='fontFamily' [toolbarSettings]='toolbarSettings'
[cssClass]='cssClass'>
<ng-template #valueTemplate>
```

```

    <p>The Rich Text Editor triggers events based on its actions. </p>
    <p> The events can be used as an extension point to perform custom
    operations.</p>
    <ul>
      <li>created - Triggers when the component is rendered.</li>
      <li>change - Triggers only when RTE is blurred and changes are done to
    the content.</li>
      <li>focus - Triggers when RTE is focused in.</li>
      <li>blur - Triggers when RTE is focused out.</li>
      <li>actionBegin - Triggers before command execution using toolbar items
    or executeCommand method.</li>
      <li>actionComplete - Triggers after command execution using toolbar
    items or executeCommand method.</li>
      <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService
]
})
export class AppComponent {
  @ViewChild('sample') public rteObj?: RichTextEditorComponent;
  public fontFamily: Object = {
    default: "Noto Sans", // to define default font-family
    items: [
      {text: "Segoe UI", value: "Segoe UI", class: "e-segoe-ui", command:
"Font", subCommand: "FontName"},
      {text: "Noto Sans", value: "Noto Sans", command: "Font", subCommand:
"FontName"},
      {text: "Impact", value: "Impact,Charcoal,sans-serif", class: "e-impact",
command: "Font", subCommand: "FontName"},
      {text: "Tahoma", value: "Tahoma,Geneva,sans-serif", class: "e-tahoma",
command: "Font", subCommand: "FontName"},
    ]
  };
  public toolbarSettings: Object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough','|',
'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
'LowerCase', 'UpperCase', '|',
'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
'Outdent', 'Indent', '|',
'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public cssClass: String = "customClass";
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Check image size in Angular Rich text editor component

By using the Rich text editor's `imageUploading` event, you can get the image size before uploading and restrict the image to upload, when the given image size is greater than the allowed size.

In the following, we have validated the image size before uploading and determined whether the image has been uploaded or not.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
import { UploadingEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample
[insertImageSettings]='insertImageSettings'
[toolbarSettings]='toolbarSettings'
(imageUploading)='onImageUploading($event)'>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService
]
})
export class AppComponent {
  @ViewChild('sample') public rteObj?: RichTextEditorComponent;
  public toolbarSettings: Object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
      'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
      'LowerCase', 'UpperCase', '|',
      'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
      'Outdent', 'Indent', '|',
      'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
      'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public insertImageSettings: object = {
    saveUrl: "https://aspnetmvc.syncfusion.com/services/api/uploadbox/Save",
    path: "../Images/"
  };
  public onImageUploading = (args: UploadingEventArgs) => {
    console.log("file is uploading");
    let imgSize: number = 500000;
    let sizeInBytes: number = args.fileData.size;
    if (imgSize < sizeInBytes) {
      args.cancel = true;
    }
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize placeholder style in Angular Rich text editor component

By using `e-rte-placeholder` class, you can customize the placeholder style.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample [placeholder]='placeholder'></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
export class AppComponent {
  public placeholder: String = 'Type Something';
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize shortcut keys in Angular Rich text editor component

It can be achieved by using `formatter` property. You need to create `customformatterModel` to configure the `keyConfig` using `IHtmlFormatterModel` class and assign the same to the `formatter` property. Here, `ctrl+q` is configured to open the `Insert Hyperlink` dialog.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component } from '@angular/core';
import { ToolbarService, LinkService, ImageService } from '@syncfusion/ej2-angular-richtexteditor';
```

```

import { HtmlEditorService, IHtmlFormatterModel, HTMLFormatter } from
 '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample
[formatter]='formatter'>
    <ng-template #valueTemplate>
      <p>The Rich Text Editor triggers events based on its actions. </p>
      <p>The events can be used as an extension point to perform custom
operations.</p>
      <ul>
        <li>created - Triggers when the component is rendered.</li>
        <li>change - Triggers only when RTE is blurred and changes are done
to the content.</li>
        <li>focus - Triggers when RTE is focused in.</li>
        <li>blur - Triggers when RTE is focused out.</li>
        <li>actionBegin - Triggers before command execution using toolbar
items or executeCommand method.</li>
        <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
        <li>destroyed - Triggers when the component is destroyed.</li>
      </ul>
    </ng-template>
  </ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
export class AppComponent {
  public customHTMLModel: IHtmlFormatterModel = { // formatter is used to
configure the custom key
  keyConfig: {
    'insert-link': 'ctrl+q', // confite the desired key
  }
};
  public formatter: any = new HTMLFormatter(this.customHTMLModel); // to
configure custom key
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You need to import `IHtmlFormatterModel` and `HTMLFormatter` to configure the shortcut key.

Set the cursor at the specific range in Angular Rich text editor component

This can be achieved by using `setRange` method in the RTE using `NodeSelection` instance. In this below sample, you have passed the text node (specific location in RTE content) in `setStart` method and passed the range in `setRange` method of RTE.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, NodeSelection, ImageService,
HtmlEditorService, RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample
(created)="onCreate()">
  <ng-template #valueTemplate>
    <p>The Rich Text Editor is WYSIWYG ("what you see is what you get")
editor useful to create and edit content, and return the valid <a
href="https://ej2.syncfusion.com/home/" target="_blank">HTML markup</a> or <a
href="https://ej2.syncfusion.com/home/"
target="_blank">markdown</a> of the content</p>
    <p id="key"><b>Toolbar</b></p>
    <ol>
      <li>
        <p>Toolbar contains commands to align the text, insert link,
insert image, insert list, undo/redo operations, HTML view, etc </p>
      </li>
      <li>
        <p>Toolbar is fully customizable </p>
      </li>
    </ol>
  </ng-template>
</ejs-richtexteditor>
    <div><input id="btn" class="e-btn" type="button" value="Set cursor point"
/></div>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
export class AppComponent {
  @ViewChild('sample') public rteObj?: RichTextEditorComponent;
  onCreate() {
    const element: Element = (this.rteObj!.contentModule as
any).getDocument().getElementById("key");
    document.getElementById('btn')!.onclick = function() {
      const selectioncursor: NodeSelection = new NodeSelection();
      const range: Range = document.createRange();
      range.setStart((element.childNodes[0] as any).firstChild,
element.textContent!.length); // to set the range
```

```
selectioncursor.setRange(document, range); // to set the cursor
};
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Update value in Angular Rich text editor component

To achieve this, you need to bind the **keydown** event to the RTE content and capture the **ctrl + s** key press using its **keyCode**.

In the **keydown** event handler, the **updateValue** method is called to update the **value** property and then you can save the content in the required database using the same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService,
RichTextEditorComponent } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [

    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample [(value)]='value'
(created)="onCreate()"></ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
export class AppComponent {
  @ViewChild('sample') public rteObj?: RichTextEditorComponent;
  public value: string = `
<p>The Rich Text Editor triggers events based on its actions. </p>
<p> The events can be used as an extension point to perform custom
operations.</p>
<ul>
  <li>created - Triggers when the component is rendered.</li>
  <li>change - Triggers only when RTE is blurred and changes are done to
the content.</li>
  <li>focus - Triggers when RTE is focused in.</li>
  <li>blur - Triggers when RTE is focused out.</li>
  <li>actionBegin - Triggers before command execution using toolbar items
or executeCommand method.</li>
  <li>actionComplete - Triggers after command execution using toolbar items
or executeCommand method.</li>
```

```

    <li>destroyed - Triggers when the component is destroyed.</li>
</ul>`;
onCreate() {
    const instance: any = this.rteObj;
    instance.contentModule.getDocument().addEventListener("keydown",
function(e: any): void {
    if (e.key === 's' && e.ctrlKey === true) {
        e.preventDefault(); // to prevent default ctrl+s action
        instance.updateValue(); // to update the value after editing
        let value: any = instance.value; // you can get the RTE content to
save in the desired database
    }
    });
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Rename images in server in Angular Rich text editor component

By using the [insertImageSettings](#) property, you can specify the server handler to upload the selected image. Then, you can bind the [imageUploadSuccess](#) event, to receive the modified file name from the server and update it in the Rich Text Editor's insert image dialog.

`HTML

```

<ejs-richtexteditor id='defaultRTE' #sample [insertImageSettings]='insertImageSettings'
[toolbarSettings]='toolbarSettings' (imageUploadSuccess)='onImageUploadSuccess($event)'>
<ng-template #valueTemplate>
<p>The Rich Text Editor triggers events based on its actions. </p>
<p> The events can be used as an extension point to perform custom operations.</p>
<ul>
<li>created - Triggers when the component is rendered.</li>
<li>change - Triggers only when Rich Text Editor is blurred and changes are done to the content.</li>
<li>focus - Triggers when Rich Text Editor is focused in.</li>
<li>blur - Triggers when Rich Text Editor is focused out.</li>
<li>actionBegin - Triggers before command execution using toolbar items or executeCommand
method.</li>
<li>actionComplete - Triggers after command execution using toolbar items or executeCommand
method.</li>
<li>destroyed – Triggers when the component is destroyed.</li>
</ul>

```



```

</ng-template>
</ejs-richtexteditor>
`typescript
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService, RichTextEditorComponent } from
 '@syncfusion/ej2-angular-richtexteditor';
@Component({
  selector: 'app-root',
  templateUrl: 'app.compoenent.html',
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
export class AppComponent {
  @ViewChild('sample') public rteObj: RichTextEditorComponent;
  public toolbarSettings: Object = {
    items: ['Bold', 'Italic', 'Underline', 'StrikeThrough', '|',
    'FontName', 'FontSize', 'FontColor', 'BackgroundColor',
    'LowerCase', 'UpperCase', '|',
    'Formats', 'Alignments', 'OrderedList', 'UnorderedList',
    'Outdent', 'Indent', '|',
    'CreateLink', 'Image', '|', 'ClearFormat', 'Print',
    'SourceCode', 'FullScreen', '|', 'Undo', 'Redo']
  };
  public insertImageSettings: object = {
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Rename",
    path: "../Images/"
  };
  public onImageUploadSuccess = (args: any) => {
    if (args.e.currentTarget.getResponseHeader('name') != null) {
      args.file.name = args.e.currentTarget.getResponseHeader('name');
      let filename: any = document.querySelectorAll(".e-file-name")[0];
      filename.innerHTML = args.file.name.replace(document.querySelector(".e-file-type")[0].innerHTML,
        "");
      filename.title = args.file.name;
    }
  }
}

```

```
}  
}  
,
```

To configure the server-side handler, refer the below code.

```
`csharp  
[AcceptVerbs("Post")]  
public void Rename()  
{  
try  
{  
var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];  
imageFile = httpPostedFile.FileName;  
if (httpPostedFile != null)  
{  
var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Images");  
if (!Directory.Exists(fileSave))  
{  
Directory.CreateDirectory(fileSave);  
}  
var fileName = Path.GetFileName(httpPostedFile.FileName);  
var fileSavePath = Path.Combine(fileSave, fileName);  
while (System.IO.File.Exists(fileSavePath))  
{  
imageFile = "rtelImage" + x + "-" + fileName;  
fileSavePath = Path.Combine(fileSave, imageFile);  
x++;  
}  
if (!System.IO.File.Exists(fileSavePath))  
{  
httpPostedFile.SaveAs(fileSavePath);  
HttpResponse Response = System.Web.HttpContext.Current.Response;  
Response.Clear();  
Response.Headers.Add("name", imageFile);
```

```

Response.ContentType = "application/json; charset=utf-8";
Response.StatusDescription = "File uploaded succesfully";
Response.End();
}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
    Response.StatusDescription = e.Message;
    Response.End();
}
}
`

```

File attachment in Angular Rich text editor component

The Rich Text Editor allows you to attach a file based on the file upload. You can attach your files using the file upload or drag-and-drop from your local path. When the file upload gets success, the attachment link inserts into the content.

In the below sample, configure the saveUrl and path properties to achieve file attachments.

1. saveUrl: Provides service URL to save the files.
2. path: Specifies the location to store the image.

The following sample illustrates how to attach a file in the RichTextEditor.

```

`html
<ejs-richtexteditor id='defaultRTE' #sample [insertImageSettings]='insertImageSettings'>
<ng-template #valueTemplate>
<p>The Rich Text Editor triggers events based on its actions. </p>
<p> The events can be used as an extension point to perform custom operations.</p>
<ul>
<li>created - Triggers when the component is rendered.</li>

```

change - Triggers only when Rich Text Editor is blurred and changes are done to the content.

focus - Triggers when Rich Text Editor is focused in.

blur - Triggers when Rich Text Editor is focused out.

actionBegin - Triggers before command execution using toolbar items or executeCommand method.

actionComplete - Triggers after command execution using toolbar items or executeCommand method.

destroyed – Triggers when the component is destroyed.

</ng-template>

</ejs-richtexteditor>

<ejs-uploader #defaultupload id='defaultfileupload' [asyncSettings]='path' [dropArea]='dropElement' (success)='onImageUploadSuccess(\$event)'></ejs-uploader>

,

`typescript

```
import { Component, ViewChild } from '@angular/core';
```

```
import { ToolbarService, LinkService, ImageService, HtmlEditorService, RichTextEditorComponent ,
NodeSelection } from '@syncfusion/ej2-angular-richtexteditor';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: 'app.compoenent.html',
```

```
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService , NodeSelection ]
```

```
})
```

```
export class AppComponent {
```

```
  @ViewChild('sample') public rteObj: RichTextEditorComponent;
```

```
  @ViewChild('defaultupload') public uploadObj: uploaderComponent;
```

```
  public insertImageSettings: object = {
```

```
    saveUrl: "[SERVICEHOSTEDPATH]/api/uploadbox/Save",
```

```
    path: "../Files/"
```

```
  };
```

```
  public path: Object = {
```

```
    saveUrl: '[SERVICEHOSTEDPATH]/api/uploadbox/Save',
```

```
  };
```

```
  public dropElement ='#defaultRTE'
```

```

public selection: NodeSelection = new NodeSelection();
public range: Range;
public saveSelection: NodeSelection;
public onImageUploadSuccess = (args: any) => {
    this.rteObj.contentModule.getEditPanel().focus();
    this.range = this.selection.getRange(document);
    this.saveSelection = this.selection.save(this.range, document);
    var fileUrl = document.URL + this.rteObj.insertImageSettings.path + args.file.name;
    if (rteObj.formatter.getUndoRedoStack().length === 0) {
        rteObj.formatter.saveData();
    }
    saveSelection.restore();
    rteObj.executeCommand('createLink', { url: fileUrl, text: fileUrl, selection: saveSelection });
    rteObj.formatter.saveData();
    rteObj.formatter.enableUndo(rteObj);
    this.uploadObj.clearAll();
}
}
,

```

To config server-side handler, refer the below code.

```

` csharp
int x = 0;
string file;
[AcceptVerbs("Post")]
public void Save()
{
    try
    {
        var httpPostedFile = System.Web.HttpContext.Current.Request.Files["UploadFiles"];
        file = httpPostedFile.FileName;
        if (httpPostedFile != null)
        {
            Console.WriteLine(System.Web.HttpContext.Current.Server.MapPath("~/Files"));

```

```
var fileSave = System.Web.HttpContext.Current.Server.MapPath("~/Files");
if (!Directory.Exists(fileSave))
{
    Directory.CreateDirectory(fileSave);
}
var fileName = Path.GetFileName(httpPostedFile.FileName);
var fileSavePath = Path.Combine(fileSave, fileName);
while (System.IO.File.Exists(fileSavePath))
{
    file = "rte" + x + "-" + fileName;
    fileSavePath = Path.Combine(fileSave, file);
    x++;
}
if (!System.IO.File.Exists(fileSavePath))
{
    httpPostedFile.SaveAs(fileSavePath);
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.Headers.Add("name", file);
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusDescription = "File uploaded succesfully";
    Response.Headers.Add("url", fileSavePath);
    Response.End();
}
}
}
catch (Exception e)
{
    HttpResponseMessage Response = System.Web.HttpContext.Current.Response;
    Response.Clear();
    Response.ContentType = "application/json; charset=utf-8";
    Response.StatusCode = 204;
    Response.Status = "204 No Content";
}
```

```
Response.StatusDescription = e.Message;
```

```
Response.End();
```

```
}
```

```
}
```

```
,
```

Format code block in Angular Rich text editor component

You can configure code block formatting as a separate toolbar button by adding the **InsertCode** keyword within the [toolbarSettings](#) items property.

The InsertCode button has a toggle state to apply code block formatting to the editor and remove code block formatting from the editor.

The following sample demonstrates how to config the InsertCode button in toolbar and set the background color to “pre” tag for highlighting the code block.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RichTextEditorAllModule } from '@syncfusion/ej2-angular-richtexteditor'
import { Component, ViewChild } from '@angular/core';
import { ToolbarService, LinkService, ImageService, HtmlEditorService } from '@syncfusion/ej2-angular-richtexteditor';
@Component({
  imports: [
    RichTextEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-richtexteditor id='defaultRTE' #sample
[toolbarSettings]='tools'>
  <ng-template #valueTemplate>
    <p>The Rich Text Editor triggers events based on its actions. </p>
    <p> The events can be used as an extension point to perform custom
operations.</p>
    <ul>
      <li>created - Triggers when the component is rendered.</li>
      <li>change - Triggers only when RTE is blurred and changes are done to
the content.</li>
      <li>focus - Triggers when RTE is focused in.</li>
      <li>blur - Triggers when RTE is focused out.</li>
      <li>actionBegin - Triggers before command execution using toolbar items
or executeCommand method.</li>
      <li>actionComplete - Triggers after command execution using toolbar
items or executeCommand method.</li>
      <li>destroyed - Triggers when the component is destroyed.</li>
    </ul>
  </ng-template>
</ejs-richtexteditor>`,
  providers: [ToolbarService, LinkService, ImageService, HtmlEditorService ]
})
```

```
export class AppComponent {  
  public tools: object = {  
    items: ['InsertCode']  
  };  
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';  
import { AppComponent } from './app.component';  
import 'zone.js';  
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Schedule

Getting started with Angular Schedule component

This section briefly explains how to create [Link to the Video](#) component and configure its available functionalities in Angular Environment.

To get start quickly with Angular Scheduler using CLI and Schematics, you can check on this video:

Setup Angular Environment

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

To install Angular CLI use the following command.

```
`bash
```

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

```
`
```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
```

```
ng new my-app
```

```
cd my-app
```

```
`
```

Installing Syncfusion Schedule package

Syncfusion packages are distributed in npm as `@syncfusion` scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(>=20.2.36) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-schedule](#) package to the application.

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

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-schedule@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-schedule@ngcc --save
`
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-schedule:"20.2.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Registering Schedule Module

Import Schedule module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-schedule` [`src/app/app.module.ts`].

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the ScheduleModule for the Schedule component
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { AppComponent } from './app.component';
@NgModule({
//declaration of ej2-angular-schedule module into NgModule
imports: [ BrowserModule, ScheduleModule ],
declarations: [ AppComponent ],
bootstrap: [ AppComponent ]
})
```

```

})
export class AppModule { }
`

```

Adding CSS reference

Schedule CSS files are available in the ej2-angular-schedule package folder. This can be referenced in your application using the following code.

```
[src/styles/styles.css].
```

```

`css
@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-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-schedule/styles/material.css';
`

```

Module injection

Each view types available in scheduler are maintained as individual modules and to work with those views, it is necessary to inject the required modules. The following modules are available in scheduler namely,

- **Day** - Inject this module for displaying day view.
- **Week** - Inject this module for displaying week view.
- **WorkWeek** - Inject this module for displaying work week view.
- **Month** - Inject this module for displaying month view.
- **Year** - Inject this module for displaying year view.
- **Agenda** - Inject this module for displaying agenda view.
- **MonthAgenda** - Inject this module for displaying month agenda view.
- **TimelineViews** - Inject this module for displaying timeline day, timeline week, timeline work week view.
- **TimelineMonth** - Inject this module for displaying timeline month view.
- **TimelineYear** - Inject this module for displaying timeline year view.

These modules should be injected into the schedule using the `providers` method within the `app.component.ts` file as shown below. On doing so, only the injected views will be loaded and displayed on the schedule.

```
[src/app/app.component.ts]
```

```
`typescript
@Component({
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService,
    MonthAgendaService, TimelineViewsService, TimelineMonthService]
})
`
```

Initialize the Schedule

Modify the template in [src/app/app.component.ts] file to render the `ej2-angular-schedule` component.

[src/app/app.component.ts]

```
`typescript
import { Component } from '@angular/core';

import { DayService, WeekService, WorkWeekService, MonthService, AgendaService } from
  '@syncfusion/ej2-angular-schedule';

@Component({
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService],
  // specifies the template string for the Schedule component
  template: <ejs-schedule> </ejs-schedule>
})
export class AppComponent { }
`
```

Now, run the application in the browser using the following command.

```
`sh
npm start
`
```

The output will display the empty scheduler.

Populating 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. To define any appointments, start and end time fields are mandatory. In the following example, you can see the appointment defined with default fields such as Id, Subject, StartTime and EndTime.

[src/app/app.component.ts]

```
`typescript
import { Component } from '@angular/core';
```

```
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
@Component({
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: <ejs-schedule [eventSettings]='eventSettings'></ejs-schedule>
})
export class AppComponent {
  public data: object[] = [{
    Id: 1,
    Subject: 'Meeting',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30)
  }];
  public eventSettings: EventSettingsModel = {
    dataSource: this.data
  }
}
```

You can also provide different names to these default fields, for which the custom names of those fields must be mapped appropriately within `fields` property as shown below.

```
`typescript
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService, MonthService,
  AgendaService } from '@syncfusion/ej2-angular-schedule';
@Component({
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' [selectedDate]='selectedDate'
    [eventSettings]='eventSettings'></ejs-schedule>`
})
export class AppComponent {
  public data: object [] = [{
    id: 2,
```

```

eventName: 'Meeting',
startTime: new Date(2018, 1, 15, 10, 0),
endTime: new Date(2018, 1, 15, 12, 30),
isAllDay: false
});
public selectedDate: Date = new Date(2018, 1, 15);
public eventSettings: EventSettingsModel = {
  dataSource: this.data,
  fields: {
    id: 'id',
    subject: { name: 'eventName' },
    isAllDay: { name: 'isAllDay' },
    startTime: { name: 'startTime' },
    endTime: { name: 'endTime' },
  }
};
}
`

```

The other fields available in Scheduler can be referred from [here](#).

Setting date

Scheduler usually displays the system date as its current date. To change the current date of Scheduler with specific date, define the `selectedDate` property.

```
[src/app/app.component.ts]
```

```

`typescript
import { Component } from '@angular/core';
import { DayService, WeekService, WorkWeekService, MonthService, AgendaService } from
 '@syncfusion/ej2-angular-schedule';
@Component({
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService],
  // specifies the template string for the Schedule component
  template: <ejs-schedule width='100%' height='550px' [selectedDate]='selectedDate'></ejs-
  schedule>
})
`

```

```
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
}
,
```

Setting view

Scheduler displays **week** view by default. To change the current view, define the applicable view name to the **currentView** property. The applicable view names are,

- Day
- Week
- WorkWeek
- Month
- Year
- Agenda
- MonthAgenda
- TimelineDay
- TimelineWeek
- TimelineWorkWeek
- TimelineMonth
- TimelineYear

`typescript

```
import { Component } from '@angular/core';

import { DayService, WeekService, WorkWeekService, MonthService, AgendaService, View } from
 '@syncfusion/ej2-angular-schedule';

@Component({
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService],
  // specifies the template string for the Schedule component
  template: <ejs-schedule width='100%' height='550px' [selectedDate]='selectedDate'
 [currentView]='currentView' ></ejs-schedule>
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public currentView: View = 'Month';
}
,
```

Individual view customization

Each individual Scheduler views can be customized with its own options such as setting different start and end hour on Week and Work Week views, whereas hiding the weekend days on Month view alone.

This can be achieved by defining views property to accept the array of object type, where each object depicts the individual view customization.

The output will display the Scheduler with the specified view configuration.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule, View } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { WeekService, MonthService, WorkWeekService, EventSettingsModel }
from '@syncfusion/ej2-angular-schedule';
import { defaultData } from './datasource';
@Component({
  imports: [

    ScheduleModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [WeekService, MonthService, WorkWeekService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"><e-views> <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]="showWeekend"></e-view></e-views></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public showWeekend: boolean = false;
  public eventSettings: EventSettingsModel = { dataSource: defaultData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

You can also explore our [Angular Scheduler example](#) that shows how to use the toolbar buttons to play with Scheduler functionalities.

Module injection in Angular Schedule component

The crucial step on creating a Scheduler with required views, is to import and inject the required modules. The modules that are available on Scheduler to work with its related functionalities are as follows.

- `DayService` - Inject this module to work with day view.
- `WeekService` - Inject this module to work with week view.
- `WorkWeekService` - Inject this module to work with work week view.
- `MonthService` - Inject this module to work with month view.
- `YearService` - Inject this module to work with year view.
- `AgendaService` - Inject this module to work with agenda view.
- `MonthAgendaService` - Inject this module to work with month agenda view.
- `TimelineViewsService` - Inject this module to work with timeline day, timeline week, and timeline work week views.
- `TimelineMonthService` - Inject this module to work with timeline month view.
- `TimelineYearService` - Inject this module to work with timeline year view.
- `DragAndDropService` - Inject this module to allow drag and drop of appointments on Scheduler.
- `ResizeService` - Inject this module for enabling the resize functionality of appointments on Scheduler.
- `ExcelExportService` - Inject this module for exporting the Scheduler events data as excel file format.
- `ICalendarExportService` - Inject this module for exporting the Scheduler events data to an ICS file.
- `ICalendarImportService` - Inject this module for importing the Scheduler events data from an ICS file.

Module injection

The required modules should be injected into the Scheduler using the `@NgModule.providers` section within the `app.component.ts` file as shown below. On doing so, only the injected module functionalities will be loaded and can be worked with Scheduler.

[src/app/app.component.ts]

```
`typescript
@NgModule({
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService,
    MonthAgendaService]
})
```

Note: If a Scheduler `currentView` is set to any one of the available views without injecting that respective view module, then a script error will occur and the Scheduler will not render.

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Scheduler interactions in Angular Schedule component

The following table describes the Scheduler actions and also illustrates how those actions are carried out through mouse and touch interactions on Scheduler.

| Actions | Mouse interaction | Touch interaction |

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

| Single click or tap on cells | Single click on a cell to select a cell. | Single tapping on cells, will display a + icon on the cell. Tapping on it again will open the new event editor window. |

| Multiple cell selection | Single click on a cell and drag the selection to other cells to enable multiple cell selection. | No multiple cell selection is allowed using touch gestures. |

| Event selection | Single click on an event to select it. | Tap holding on events, select an event and opens a small popup at the top holding the options to edit or delete. The popup also displays the selected event's subject. |

| Multiple event selection and deletion | Pressing **Ctrl** key and altogether single clicking on multiple events one after the other will enable multiple event selection. Pressing **Delete** key after event selection will delete all the selected events. | Tap hold an event to select it, which opens a small popup at the top holding the options to edit or delete. As a continuation of this action, keep on single tapping on other events, to enable multiple event selection. Also, the popup displayed at the top remains in opened state, showing the count of the number of selected events. Pressing **Delete** option from the popup will delete all the selected events. |

| Date navigation | Clicking on the previous or next date navigation icons in the header bar allows you to navigate between dates. | Swiping the scheduler view port to the left or right will allow you to navigate between the dates on touch devices. NOTE: Swiping does not work when horizontal scroller present in the Scheduler. You can also make use of the previous and next navigation icons at the header bar to navigate. |

| View navigation | Click on an event and try moving it over the Scheduler to enable drag and drop action. | The view options are available within the popup options at the top right extreme end of the header bar and you can choose the view from it. |

| Drag and drop | Click on an event and try moving it over the Scheduler to enable drag and drop action. | Tap hold the event and try moving it over the Scheduler to enable drag and drop action. |

| Event resizing | Hover the mouse across the extremities or edges of the Scheduler events and when the mouse pointer changes into resize handler, now click and start resizing an event to the desired time range. | Touch the event extremities and start resizing the events directly. |

| Tooltip | Hover the mouse pointer over the events or resource header and the tooltip will be displayed. | Tap holding the events will open the tooltip on events. |

| Open editor window | Double click on cells or events to open the editor window. | Double click on cells or events to open the editor window. Single tap on cells, which displays a + icon on the cell. Now, tap on it again to open the new event editor window. To open the editor on events, single tap on it and then click on the edit icon to open the editor window in **Edit** mode. |

| Open quick info popup | Single clicking on a cell will open a quick popup prompting for new event creation. Single clicking on an event will open a popup displaying event information along with the option to edit and delete it. | No quick info popup is available while single tapping on cells. Single tapping on events, opens the popup showing event information. |

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler exampleLink to the Video](#) to know how to present and manipulate data.

Appointments in Angular Schedule component

Appointments can be anything that are scheduled for a specific time period. It can be created on varied time range and each appointment is categorized based on this range. The Scheduler events can be categorized as,

- Normal events
- Spanned events
- All-day events
- Recurring events

To have a quick glance on how to add appointments to the Angular Scheduler and some of its advanced event-handling options, watch this video:

Normal events

Represents an appointment that is created for any specific time interval within a day.

Creating a normal event

The following example depicts how to define a normal event on the Scheduler, with event data being loaded from simple JSON data.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
TimelineMonthService, MonthService, AgendaService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, MonthService, AgendaService,
TimelineMonthService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" ></ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    Id: 2,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30)
```

```

    }];
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: this.data
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Spanned events

Represents an appointment that is created for more than 24 hours, and usually displayed on the all-day row. Also, represents another type of appointment that is created for more than one day but less than 24 hours, and usually displayed appropriately on both the days.

For example, if an appointment is created for two days say from November 25, 2018 – 11.00 PM to November 26, 2018 2.00 AM but less than 24 hours time interval, then the appointment is split into two partitions and will be displayed on both the days.

All-day events

Represents an appointment that is created for an entire day such as holiday events. It is usually displayed separately in an all-day row, a separate row for all-day appointments below the date header section. In Timeline views, the all-day appointments displays in the working space area, and no separate all-day row is present in that view.

To change normal appointment into all-day event, set `isAllDay` field to true.

Hide all-day row events

You can make use of the CSS customization to prevent the display of all-day row appointments on the Scheduler UI.

```

`typescript
<style>
.e-schedule .e-date-header-wrap .e-schedule-table thead {
display: none;
}
</style>
`

```

You can also enable scroller for all-day row, please [refer](#) here to know more.

Expand all day appointments view on initial load

When you have larger number of appointments in all-day view, you can show all all-day events using `dataBound` event on at initial load. So, user don't have to click the toggle to expand all-day events.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    ScheduleComponent,
    DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    ResizeService,
    DragAndDropService,
    MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
let initialLoad = true;
@Component({
    imports: [

        ScheduleModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="control-section">
        <div class="col-lg-12 content-wrapper">
            <ejs-schedule #scheduleObj width='100%' height='750px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dataBound)="dataBound($event)" ></ejs-
schedule>
        </div>
    </div>`,
    encapsulation: ViewEncapsulation.None,
    providers: [
        DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        ResizeService,
        DragAndDropService,
        MonthAgendaService
    ],
})
export class AppComponent {
    @ViewChild('scheduleObj') public scheduleObj?: ScheduleComponent;
    dataBound(eventData: any) {
        if (initialLoad) {
            let elements:HTMLInputElement=this.scheduleObj?.element.querySelector('.e-
all-day-appointment-section') as HTMLInputElement;
            elements.click();
            initialLoad = false;
        }
    }
    public data: object[] = [
        {

```

```

        EndTime: new Date(2022, 4, 4, 0, 0),
        Id: '1',
        IsAllDay: true,
        StartTime: new Date(2022, 4, 2, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Install Bullnose Brick/ Coping
| Jones | 3521',
    },
    {
        EndTime: new Date(2022, 3, 30, 0, 0),
        Id: '2',
        IsAllDay: true,
        StartTime: new Date(2022, 3, 29, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Plumbing Checklist | Jaimungal
| 3671 :: Pool',
    },
    {
        EndTime: new Date(2022, 4, 7, 0, 0),
        Id: '3',
        IsAllDay: true,
        StartTime: new Date(2022, 4, 2, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Waterline Tile | Jaimungal |
3671 :: Pool',
    },
    {
        EndTime: new Date(2022, 3, 30, 0, 0),
        Id: '4',
        IsAllDay: true,
        StartTime: new Date(2022, 3, 28, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Underground Plumbing |
Jaimungal | 3671 :: Pool',
    },
    {
        EndTime: new Date(2022, 4, 4, 0, 0),
        Id: '5',
        IsAllDay: true,
        StartTime: new Date(2022, 4, 3, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Backfill/ Compaction |
Jaimungal | 3671 :: Pool',
    },
    {
        EndTime: new Date(2022, 4, 7, 0, 0),
        Id: '6',
        IsAllDay: true,
        StartTime: new Date(2022, 4, 4, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Install Bullnose Brick/ Coping
| Jaimungal | 3671 :: Pool',
    },
    {
        EndTime: new Date(2022, 4, 1, 0, 0),
        Id: '7',
        IsAllDay: true,

```

```

        StartTime: new Date(2022, 3, 30, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Steel/ Checklist | VP Highland
Model | 3719 :: Pool',
    },
    {
        Description:
            'Let Yves know I did not see skimmer southern gunite did shell',
        EndTime: new Date(2022, 4, 4, 0, 0),
        Id: '8',
        IsAllDay: true,
        StartTime: new Date(2022, 4, 3, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Shotcrete Shell | VP Highland
Model | 3719 :: Pool',
    },
    {
        EndTime: new Date(2022, 3, 30, 0, 0),
        Id: '9',
        IsAllDay: true,
        StartTime: new Date(2022, 3, 29, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Tile Selections/ Pavers/
Finish | VP Highland Model | 3719 :: Pool',
    },
    {
        EndTime: new Date(2022, 3, 30, 0, 0),
        Id: '10',
        IsAllDay: true,
        StartTime: new Date(2022, 3, 26, 0, 0),
        Subject:
            '<i class="fas fa-truck-pickup"></i> | Layout/ Form Rebar Shell | VP
Highland Model | 3719 :: Pool',
    },
];
public selectedDate: Date = new Date(2022, 3, 26);
public eventSettings = {
    dataSource: this.data,
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customize the rendering of the spanned events

By default, Scheduler will renders the spanned events (appointment with more than 24 hours duration) in the all-day row by setting `AllDayRow` will the default type renders to the `spannedEventPlacement` option within the `eventSettings` property. Now we can customize rendering of the that events inside the work cells itself by modifying the `spannedEventPlacement` option as `TimeSlot`. In this following example, shows how to render the spanned appointments inside the work cells as follows.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, TimelineMonthService,
MonthService, AgendaService, WorkWeekService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule';
@Component({
imports: [

    ScheduleModule,
    ButtonModule
],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, MonthService, AgendaService,
TimelineMonthService, WorkWeekService, MonthAgendaService],
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" ></ejs-
schedule>`
})
export class AppComponent {
    public data: object[] = [{
        Id: 1,
        Subject: 'Paris',
        StartTime: new Date(2018, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 17, 12, 30),
        IsAllDay: false
    }, {
        Id: 2,
        Subject: 'London',
        StartTime: new Date(2018, 1, 16, 12, 0),
        EndTime: new Date(2018, 1, 18, 13, 0),
        IsAllDay: false
    }
    ];
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: this.data,
        spannedEventPlacement: 'TimeSlot'
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Recurring events

Represents an appointment that is created for a certain time interval and occurring repeatedly on a daily, weekly, monthly or yearly basis at the same time interval based on the provided recurrence rule. Usually, the recurring events are indicated by a repeat marker added at the bottom-right position.

Creating a recurring event

The following example depicts how to create a recurring event on Scheduler with the specific recurrence rule. In the following example, an event is made to repeat on daily mode and ends after 5 occurrences.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, TimelineMonthService,
MonthService, AgendaService, WorkWeekService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, TimelineMonthService, MonthService,
  AgendaService, WorkWeekService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    Id: 2,
    Subject: 'Paris',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5'
  }];
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: this.data
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```


Adding exceptions

A few instance of the recurrence series can be excluded on specific dates, by adding those exceptional dates to the `recurrenceException` field. These date values should be given in the ISO date time format with no hyphens(-) separating the date elements.

For example, 22nd February 2018 can be represented as 20180222. Also, the time part being represented in UTC format needs to add "Z" after the time portion with no space. "07:30:00 UTC" is therefore represented as "073000Z".

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
  AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    Id: 1,
    Subject: 'Paris',
    StartTime: new Date(2018, 0, 28, 10, 0),
    EndTime: new Date(2018, 0, 28, 12, 30),
    IsAllDay: false,
    RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
    RecurrenceException:
'20180129T043000Z,20180131T043000Z,20180202T043000Z'
  }];
  public selectedDate: Date = new Date(2018, 0, 28);
  public eventSettings: EventSettingsModel = {
    dataSource: this.data
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Editing an occurrence from a series

To dynamically edit a particular occurrence from an event series and display it on the initial load of Scheduler, the edited occurrence needs to be added as a new event to the dataSource collection, with an additional `recurrenceID` field defined to it. The `recurrenceID` field of edited occurrence usually maps the ID value of the parent event.

In this example, a recurring instance that displays on the date 30th Jan 2018 is edited with different timings. Therefore, this particular date is excluded from the parent recurring event that repeats from 28th January 2018 to 4th February 2018. This can be done by adding the `recurrenceException` field with the excluded date value on the parent event. Also, the edited occurrence event which is created as a new event should carry the `recurrenceID` field pointing to the parent event's `Id` value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    providers: [DayService, WeekService, WorkWeekService, MonthService,
        AgendaService, MonthAgendaService],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
    public data: object[] = [{
        Id: 1,
        Subject: 'Scrum Meeting',
        StartTime: new Date(2018, 0, 28, 10, 0),
        EndTime: new Date(2018, 0, 28, 12, 30),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=8',
        RecurrenceException: '20180130T043000Z'
    }, {
        Id: 2,
        Subject: 'Scrum Meeting',
        StartTime: new Date(2018, 0, 30, 9, 0),
        EndTime: new Date(2018, 0, 30, 10, 30),
        Description: 'Meeting time changed based on team activities.',
```

```

        RecurrenceID: 1
    }
    ];
    public selectedDate: Date = new Date(2018, 0, 28);
    public eventSettings: EventSettingsModel = {
        dataSource: this.data
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Edit only the current and following events

To edit only the current and following events enable the property `editFollowingEvents` within `eventSettings` property. The edited occurrence needs to be added as a new event to the `dataSource` collection, with an additional `followingID` field defined to it. The `followingID` field of edited occurrence usually maps the ID value of the immediate parent event.

In this example, a recurring instance that displays on the date 30th Jan 2018 and its following dates are edited with different subject. Therefore, this particular date and its following dates are excluded from the parent recurring event that repeats from 28th January 2018 to 4th February 2018. This can be done by updating the `recurrenceRule` field with the until date value on the parent event. Also, the edited events which is created as a new event should carry the `followingID` field pointing to the immediate parent event's `Id` value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component } from '@angular/core';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [

        ScheduleModule,
        ButtonModule
    ],
    standalone: true,
    selector: 'app-root',
    providers: [DayService, WeekService, WorkWeekService, MonthService,
        AgendaService, MonthAgendaService],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
        [selectedDate]="selectedDate"
        [eventSettings]="eventSettings" > </ejs-schedule>`
})

```

```

    })
    export class AppComponent {
        public data: object[] = [{
            Id: 1,
            Subject: 'Scrum Meeting',
            StartTime: new Date(2018, 0, 28, 10, 0),
            EndTime: new Date(2018, 0, 28, 12, 30),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180129T043000Z;',
        }, {
            Id: 2,
            Subject: 'Scrum Meeting - Following Edited',
            StartTime: new Date(2018, 0, 30, 10, 0),
            EndTime: new Date(2018, 0, 30, 12, 30),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;UNTIL=20180204T043000Z;',
            FollowingID: 1
        }
    ];
    public selectedDate: Date = new Date(2018, 0, 28);
    public eventSettings: EventSettingsModel = {
        dataSource: this.data,
        editFollowingEvents: true
    };
}

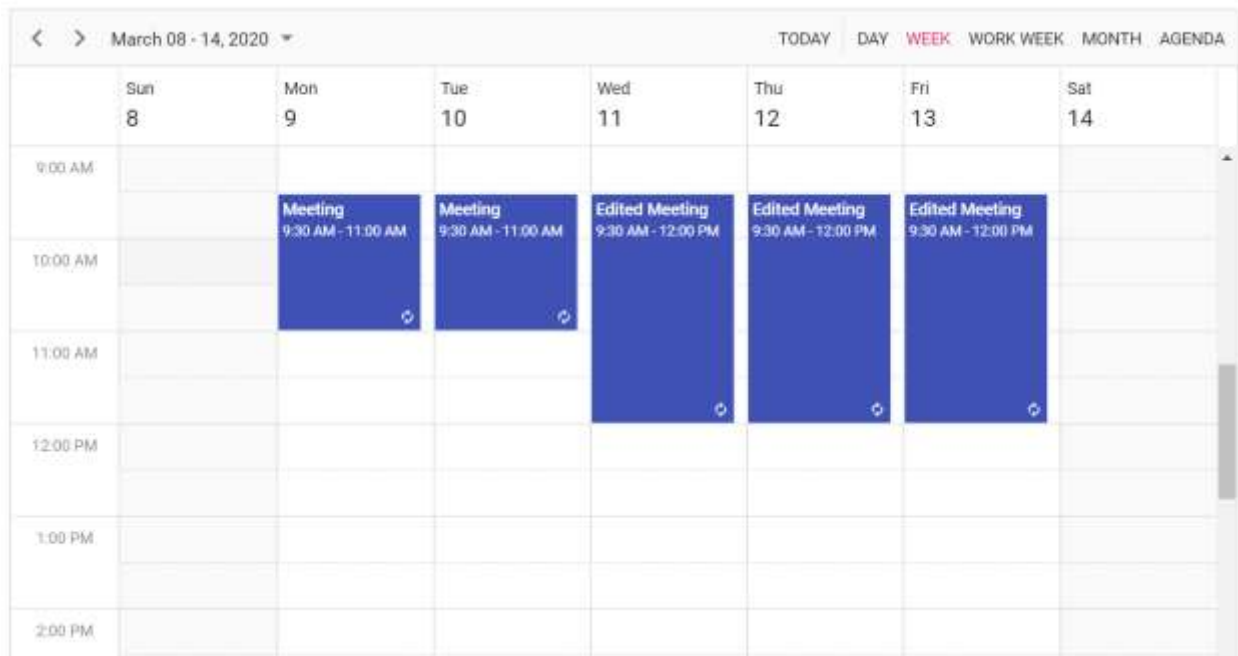
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Recurrence options and rules

Events can be repeated on a daily, weekly, monthly or yearly basis based on the recurrence rule which accepts the string value. The following details should be assigned to the `recurrenceRule` property to generate the recurring instances.

- Repeat type - daily/weekly/monthly/yearly.
- How many times it needs to be repeated?
- The interval duration.
- The time period to render the appointment, etc.

There are four repeat types available namely,

- **Daily** - Creates the recurring instances on daily basis.
- **Weekly** - Creates the recurring instances on weekly basis for the selected days.
- **Monthly** - Creates the recurring instances on monthly basis for the selected months and other provided recurrence criteria.
- **Yearly** - Creates the recurring instances on yearly basis.

Recurrence properties

The properties based on which the recurrence appointments are created with its respective time period are depicted in the following table. Also, the valid rule string can be referred from [iCalendar](#) specifications.

Refer [iCalendar](#) specifications for valid recurrence rule string.

Property	Purpose	Example
FREQ	Maintains the repeat type (Daily, Weekly, Monthly, Yearly) value of the appointment.	FREQ=DAILY;INTERVAL=1
INTERVAL	Maintains the interval value of the appointments. When you create the daily appointment at an interval of 2, the appointments are rendered on the days Monday, Wednesday and Friday (Creates an appointment on all days by leaving the interval of one day gap).	FREQ=DAILY;INTERVAL=2
COUNT	It holds the appointment's count value. When the COUNT value is 10, then 10 instances of appointments are created in the recurrence series.	FREQ=DAILY;INTERVAL=1;COUNT=10
UNTIL	This property holds the end date value (in ISO format) denoting when the recurrence actually ends.	FREQ=DAILY;INTERVAL=1;UNTIL=20180530T041343Z
BYDAY	It holds the day value(s), representing on which the appointments actually renders. Create the weekly appointment, and select the day(s) from the day options (Monday/Tuesday/Wednesday/Thursday/Friday/Saturday/Sunday). When Monday is selected, the first two letters of the selected day "MO" is saved in the BYDAY property. When multiple days are selected, the values are separated by commas.	FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE;COUNT=10
BYMONTHDAY	This property is used to store the date value of the Month, while creating the Month recurrence appointment. When you create a Monthly recurrence appointment for every 3rd day of the month, then BYMONTHDAY holds the value 3 and creates the appointment on 3rd day of every month.	FREQ=MONTHLY;BYMONTHDAY=3;INTERVAL=1;COUNT=10

| BYMONTH | This property is used to store the index value of the selected Month while creating the yearly appointments. When you create the yearly appointment on June month, the index value of June month 6 will get stored in the BYMONTH field. The appointment is created on every 6th month of a year. | `FREQ=YEARLY;BYMONTHDAY=16;BYMONTH=6;INTERVAL=1;COUNT=10` |

| BYSETPOS | This property is used to store the index value of the week. When you create the monthly appointment in second week of a month, the index value of the second week (2) is stored in BYSETPOS. | `FREQ=MONTHLY;BYDAY=MO;BYSETPOS=2;COUNT=10` |

The default recurrence related validation has been included for recurrence appointments similar to the one available in Outlook. The validation usually occurs during the recurrence appointment creation, editing, drag and drop or resizing of the recurrence appointments and also if any single occurrence changes.

Daily Frequency

| Description | Example |

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

| Daily recurring event that never ends | `FREQ=DAILY;INTERVAL=1` |

| Daily recurring event that ends after 5 occurrences | `FREQ=DAILY;INTERVAL=1;COUNT=5` |

| Daily recurring event that ends exactly on 12/12/2018 |
`FREQ=DAILY;INTERVAL=1;UNTIL=20181212T041343Z` |

| Daily event that recurs on alternative days and repeats for 10 occurrences |
`FREQ=DAILY;INTERVAL=2;COUNT=10` |

Weekly Frequency

| Description | Example |

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

| Weekly recurring event that repeats on every Monday, Wednesday and Friday and never ends |
`FREQ=WEEKLY;INTERVAL=1;BYDAY=MO,WE,FR` |

| Repeats every week Thursday and ends after 10 occurrences | `FREQ=WEEKLY;INTERVAL=1;BYDAY=TH;COUNT=10` |

| Repeats every week Monday and ends on 12/12/2018 | `FREQ=WEEKLY;INTERVAL=1;BYDAY=MO;UNTIL=20181212T041343Z` |

| Repeats on Monday, Wednesday and Friday of alternative weeks and ends after 10 occurrences |
`FREQ=WEEKLY;INTERVAL=2;BYDAY=MO,WE,FR;COUNT=10` |

Monthly Frequency

| Description | Example |

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

| Monthly recurring event that repeats on every 15th day of a month and never ends | `FREQ=MONTHLY;BYMONTHDAY=15;INTERVAL=1` |

| Monthly recurring event that repeats on every 16th day of a month and ends after 10 occurrences |
`FREQ=MONTHLY;BYMONTHDAY=16;INTERVAL=1;COUNT=10` |

| Repeats every 17th day of a month and ends on 12/12/2018 | `FREQ=MONTHLY;BYMONTHDAY=17;INTERVAL=1;UNTIL=20181212T041343Z` |

| Repeats every 2nd Friday of a month and never ends | `FREQ=MONTHLY;BYDAY=FR;BYSETPOS=2;INTERVAL=1` |

| Repeats every 4th Wednesday of a month and ends after 10 occurrences |
`FREQ=MONTHLY;BYDAY=WE; BYSETPOS=4;INTERVAL=1;COUNT=10` |

| Repeats every 4th Friday of a month and ends on 12/12/2018 |
`FREQ=MONTHLY;BYDAY=FR;BYSETPOS=4; INTERVAL=1;UNTIL=20181212T041343Z;` |

Yearly Frequency

| Description | Example |

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

| Yearly event that repeats on every 15th day of December month and never ends | `FREQ=YEARLY;BYMONTHDAY=15;BYMONTH=12;INTERVAL=1` |

| Event that repeats on every 10th day of December month and ends after 10 occurrences |
`FREQ=YEARLY; BYMONTHDAY=10;BYMONTH=12;INTERVAL=1;COUNT=10` |

| Repeats on every 12th day of December month and ends on 12/12/2025 |
`FREQ=YEARLY;BYMONTHDAY=12; BYMONTH=12;INTERVAL=1;UNTIL=20251212T041343Z` |

| Repeats on every 3rd Friday of December month and never ends |
`FREQ=YEARLY;BYDAY=FR;BYMONTH=12; BYSETPOS=3;INTERVAL=1` |

| Repeats on every 3rd Tuesday of December month and ends after 10 occurrences | `FREQ=YEARLY;BYDAY=TU;BYMONTH=12;BYSETPOS=3;INTERVAL=1;COUNT=10` |

| Repeats on every 4th Wednesday of December month and ends on 12/12/2028 |
`FREQ=YEARLY;BYDAY=WE; BYMONTH=12;BYSETPOS=4;INTERVAL=1;UNTIL=20281212T041343Z` |

Recurrence Validation

The built-in validation support has been added by default for recurring appointments during its creation, edit, drag and drop or resize action. The following are the possible validation alerts that displays on Scheduler while creating or editing the recurring events.

| Validation messages | Description |

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

| The recurrence pattern is not valid. | This alert will raise, when the selected recurrence rule value is not a valid one. For example, when you try to select the end date value (using **Until** option) for a recurring event, which occurs before the start date, an alert will popup out saying that the chosen pattern is invalid. |

| The changes made to specific instances of this series will be cancelled and those events will match the series again. | This alert will raise, when you try to edit the whole series, whose occurrence might have been already edited. For example, If there are five occurrences and one of the occurrence is already edited. Now, when you try to edit the entire series, you will get this validation alert. |

| The duration of the event must be shorter than how frequently it occurs. Shorten the duration, or change the recurrence pattern in the recurrence event editor. | This validation will occur, if the event

duration is longer than the selected frequency. For example, if you create a recurring appointment with two days duration in **Daily** frequency with no intervals set to it, you may get this alert. |

| Some months have fewer than the selected date. For these months, the occurrence will fall on the last date of the month. | When you try to create a recurring appointment on 31st of every month, where few months won't have 31 days and in this scenario, you will get this alert. |

| Two occurrences of the same event cannot occur on the same day. | This validation will occur, when you try to edit or move any single occurrence to some other date, where another occurrence of the same event is already present. |

Event fields

The Scheduler dataSource usually holds the event instances, where each of the instance includes a collection of appropriate [fields](#). It is mandatory to map these fields with the equivalent fields of database, when remote data is bound to it. When the local JSON data is bound, then the field names defined within the instances needs to be mapped with the scheduler event fields correctly.

To create an event on Scheduler, it is enough to define the **startTime** and **endTime**. Also **id** field becomes mandatory to process CRUD actions on appropriate events.

Built-in fields

The built-in fields available on Scheduler event object are as follows.

Field name	Description
id	The id field needs to be defined as mandatory and this field usually assigns a unique ID value to each of the events.
subject	The subject field is optional, and usually assigns the summary text to each of the events.
startTime	The startTime field defines the start time of an event and it is mandatory to provide it for any of the valid event objects.
endTime	The endTime field defines the end time of an event and it is mandatory to provide the end time for any of the valid event objects.
startTimezone	It maps the startTimezone field from the dataSource and usually accepts the valid IANA timezone names. It is assumed that the value provided for this field is taken into consideration while processing the startTime field. When this field is not mapped with any timezone names, then the events will be processed based on the timezone assigned to the Scheduler.
endTimezone	It maps the endTimezone field from the dataSource and usually accepts the valid IANA timezone names. It is assumed that the value provided for this field is taken into consideration while processing the endTime field. When this field is not mapped with any timezone names, then the events will be processed based on the timezone assigned to the Scheduler.
location	It maps the location field from the dataSource and the location text value will be displayed over the events.
description	It maps the description field from the dataSource and denotes the event description which is optional.

| isAllDay | The `isAllDay` field is mapped from the `dataSource` and is used to denote whether an event is created for an entire day or for specific time alone. Usually, an event with `isAllDay` field set to true will be considered as an all-day event. |

| recurrenceID | It maps the `recurrenceID` field from `dataSource` and usually holds the ID value of the parent recurrence event. This field is applicable only for the edited occurrence events. |

| recurrenceRule | It maps the `recurrenceRule` field from `dataSource` and holds the recurrence rule value in a string format. Also, it uniquely identifies whether the event belongs to a recurring type or normal ones. |

| recurrenceException | It maps the `recurrenceException` field from `dataSource` and is used to hold the collection of exception dates, on which the recurring occurrences needs to be excluded. The `recurrenceException` should be specified in UTC format. |

| isReadOnly | It maps the `isReadOnly` field from `dataSource`. It is mainly used to make specific appointments as readonly when set to true. |

| isBlock | It maps the `isBlock` field from `dataSource`. It is used to block the particular time ranges in the Scheduler and prevents the event creation on those time slots. |

Binding different field names

When the fields of event instances has the default mapping name, it is not mandatory to map them manually. If a Scheduler's `dataSource` holds the events collection with different field names, then it is necessary to map them with its equivalent field name within the `eventSettings` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    TravelId: 2,
    TravelSummary: 'Paris',
    DepartureTime: new Date(2018, 1, 15, 10, 0),
```

```

        ArrivalTime: new Date(2018, 1, 15, 12, 30),
        FullDay: false,
        Source: 'London',
        Comments: 'Summer vacation planned for outstation.',
        Origin: 'Asia/Yekaterinburg',
        Destination: 'Asia/Yekaterinburg'
    }
};
public selectedDate: Date = new Date(2018, 1, 15);
public eventSettings: EventSettingsModel = {
    dataSource: this.data,
    fields: {
        id: 'TravelId',
        subject: { name: 'TravelSummary' },
        isAllDay: { name: 'FullDay' },
        location: { name: 'Source' },
        description: { name: 'Comments' },
        startTime: { name: 'DepartureTime' },
        endTime: { name: 'ArrivalTime' },
        starttimezone: { name: 'Origin' },
        endtimezone: { name: 'Destination' }
    }
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The mapper field `id` is of string type and has no additional validation options, whereas all other fields are of `Object` type and has additional options.

Event field settings

Each field of the Scheduler events are provided with additional settings such as options to set default value, to map with appropriate data source fields, to validate every event fields and to provide label values for those fields in the event window.

Options	Description
default	Accepts the default value to the applicable fields (Subject, Location and Description), when no values are provided to them from dataSource.
name	Accepts the field name to be mapped from the dataSource fields.
title	Accepts the label values to be displayed for the fields of event editor.
validation	Defines the validation rules to be applied on the event fields within the event editor.

In following example, the Subject field in event editor will display its appropriate label as **Summary**. When no subject value is provided while saving an event, then the appointment will be saved with the default subject value as **Add Summary**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    TravelId: 2,
    TravelSummary: 'Paris',
    DepartureTime: new Date(2018, 1, 15, 10, 0),
    ArrivalTime: new Date(2018, 1, 15, 12, 30),
    Source: 'London',
    Comments: 'Summer vacation planned for outstation.'
  }];
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: this.data,
    fields: {
      id: 'TravelId',
      subject: { name: 'TravelSummary', title: 'Summary', default: 'Add
Summary' },
      location: { name: 'Source', default: 'USA' },
      description: { name: 'Comments' },
      startTime: { name: 'DepartureTime' },
      endTime: { name: 'ArrivalTime' }
    }
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Adding Custom fields

Apart from the default Scheduler fields, the user can include 'n' number of custom fields for appointments. The following code example shows how to include two custom fields namely **Status** and **Priority** within event collection. It is not necessary to bind the custom fields within the `eventSettings`. However, those additional fields can be accessed easily, for internal processing as well as from application end.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    Id: 2,
    Subject: 'Meeting',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 12, 30),
    IsAllDay: false,
    Status: 'Completed',
    Priority: 'High'
  }];
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: this.data,
    fields: {
      id: 'Id',
      subject: { name: 'Subject' },
      isAllDay: { name: 'IsAllDay' },
      startTime: { name: 'StartTime' },
      endTime: { name: 'EndTime' }
    }
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customize the order of the overlapping events

By default, the scheduler will render the overlapping events based on the start and end time. Now we can customize the order of the overlapping events based on the custom fields by using the **sortComparer** property grouped under the **eventSettings** property. The following code example shows how to sort the appointments based on the custom field as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, SortComparerFunction, MonthAgendaService } from
'@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" > </ejs-schedule>`
})
export class AppComponent {
  public data: object[] = [{
    Id: 1,
    Subject: 'Rank 1',
    StartTime: new Date(2017, 9, 29, 10, 0),
    EndTime: new Date(2017, 9, 29, 11, 30),
    IsAllDay: false,
    RankId: '1'
  }, {
    Id: 2,
    Subject: 'Rank 3',
    StartTime: new Date(2017, 9, 29, 10, 30),
    EndTime: new Date(2017, 9, 29, 12, 30),
    IsAllDay: false,
    RankId: '3'
  }, {
    Id: 3,
    Subject: 'Rank 6',
```

```

        StartTime: new Date(2017, 9, 29, 7, 0),
        EndTime: new Date(2017, 9, 29, 14, 30),
        IsAllDay: false,
        RankId: '6'
    }, {
        Id: 4,
        Subject: 'Rank 9',
        StartTime: new Date(2017, 9, 29, 11, 0),
        EndTime: new Date(2017, 9, 29, 15, 30),
        IsAllDay: false,
        RankId: '9'
    }
    ]];
    public comparerFun: SortComparerFunction = (args: Record<string, any>[])
=> {
        args.sort((event1: Record<string, any>, event2: Record<string, any>)
=> event1['RankId'].localeCompare(event2['RankId'], undefined, { numeric:
true }));
        return args;
    }
    public selectedDate: Date = new Date(2017, 9, 29);
    public eventSettings: EventSettingsModel = {
        dataSource: this.data,
        sortComparer: this.comparerFun
    };
}

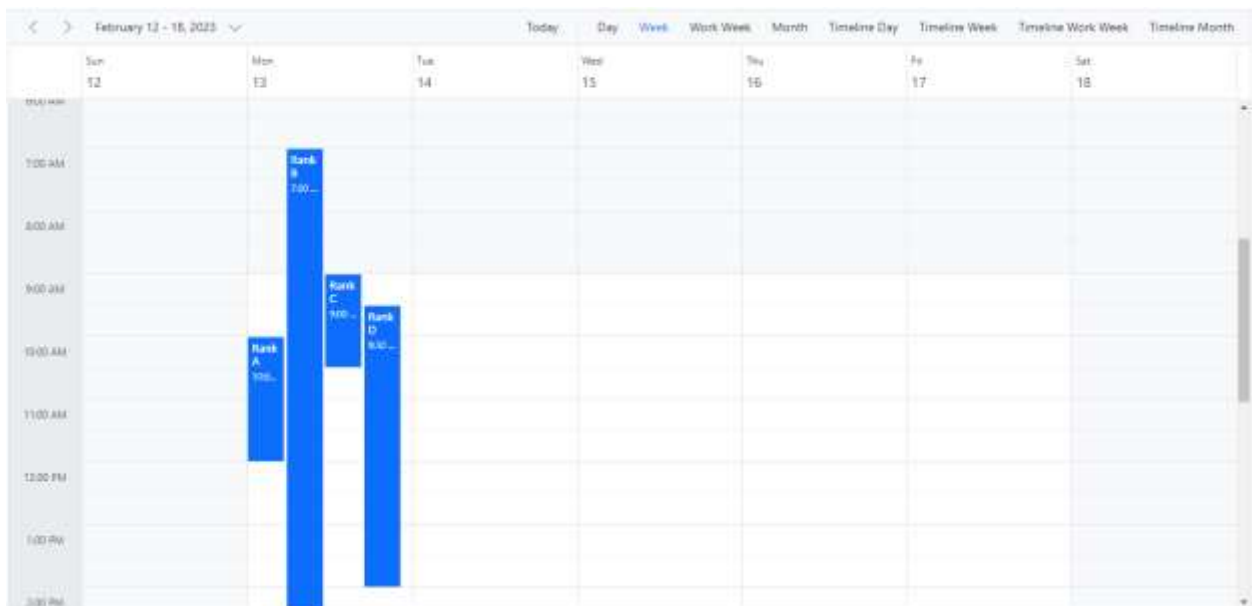
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Drag and drop appointments

Appointments can be rescheduled to any time by dragging and dropping them onto the desired location. To work with drag and drop functionality, it is necessary to inject the module `DragAndDrop` and make sure that [Link to the Video](#) is set to true on Scheduler. In mobile mode, you can drag and drop the events by tap holding an event and dropping them on to the desired location.

Learn how to drag an external item into the Angular Scheduler and the other advanced options of Drag and Resize actions from this video:

By default, drag and drop action is applicable on all Scheduler views, except Agenda, Month-Agenda and Year view.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, TimelineViewsService,
TimelineMonthService, MonthService, AgendaService, DragAndDropService,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, TimelineViewsService, TimelineMonthService,
MonthService, AgendaService, DragAndDropService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" ></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Drag and drop multiple appointments

We can drag and drop multiple appointments by enabling the `allowMultiDrag` property. We can select multiple appointments by holding the CTRL key. Once the events are selected, we can leave the CTRL key and start dragging the event.

We can also drag multiple events from one resource to another resource. In this case, if all the selected events are in the different resources, then all the events should be moved to the single resource that is related to the target event.

Note: Multiple events drag and drop is not supported on mobile devices.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, TimelineViewsService,
TimelineMonthService, MonthService, AgendaService, DragAndDropService,
WeekService, WorkWeekService, MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, TimelineViewsService, TimelineMonthService,
MonthService, AgendaService, DragAndDropService, WeekService,
WorkWeekService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [allowMultiDrag] = 'true'
[eventSettings]="eventSettings" ></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```


Disable the drag action

By default, you can drag and drop the events within any of the applicable scheduler views, and to disable it, set `false` to the `allowDragAndDrop` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, TimelineViewsService,
TimelineMonthService, MonthService, AgendaService, DragAndDropService,
WeekService, WorkWeekService, MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, TimelineViewsService, TimelineMonthService,
MonthService, AgendaService, DragAndDropService, WeekService,
WorkWeekService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [allowDragAndDrop] = 'false'
[eventSettings]="eventSettings" ></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Preventing drag and drop on specific targets

It is possible to prevent the drag action on particular target, by passing the target to be excluded in the `excludeSelectors` option within `dragStart` event. In this example, we have prevented the drag action on all-day row.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```

```

import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, DragAndDropService, DragEventArgs,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, DragAndDropService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStart) = "onDragStart($event)" ></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onDragStart(args: DragEventArgs): void {
    args.excludeSelectors = 'e-header-cells,e-header-day,e-header-date,e-
all-day-cells';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Disable scrolling on drag action

By default, while dragging an appointment to the edges, either top or bottom of the Scheduler, scrolling action takes place automatically. To prevent this scrolling, set `false` to the `scroll` value within the `dragStart` event arguments.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';

```

```

import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, DragAndDropService, DragEventArgs,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, DragAndDropService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStart) = "onDragStart($event)" ></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onDragStart(args: DragEventArgs): void {
    args.scroll = { enable: false,
      scrollBy: 30, // Specify the scroll distance if needed
      timeDelay: 500 // Specify the delay time if needed
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Controlling scroll speed while dragging an event

The speed of the scrolling action while dragging an appointment to the Scheduler edges, can be controlled within the `dragStart` event by setting the desired value to the `scrollBy` and `timeDelay` option whereas its default value is 30 minutes and 100ms.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';

```

```
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, DragAndDropService, DragEventArgs,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, DragAndDropService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStart) = "onDragStart($event)" ></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onDragStart(args: DragEventArgs): void {
    args.scroll = { enable: true, scrollBy: 5, timeDelay: 200 };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Auto navigation of date ranges on dragging an event

When an event is dragged either to the left or right extreme edges of the Scheduler and kept hold for few seconds without dropping, the auto navigation of date ranges will be enabled allowing the Scheduler to navigate from current date range to back and forth respectively. This action is set to **false** by default and to enable it, you need to set **navigation** to true within the **dragStart** event.

By default, the navigation delay is set to 2000ms. The navigation delay decides how long the user needs to drag and hold the appointments at the extremities. You can also set your own delay value for letting the users to navigate based on it, using the **timeDelay** within the **dragStart** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
```

```
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, DragAndDropService, DragEventArgs,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, DragAndDropService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStart) = "onDragStart($event)" ></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onDragStart(args: DragEventArgs): void {
    args.navigation = { enable: true, timeDelay: 4000 };
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Setting drag time interval

By default, while dragging an appointment, it moves at an interval of 30 minutes. To change the dragging time interval, pass the appropriate values to the `interval` option within the `dragStart` event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, DragAndDropService, DragEventArgs,
MonthAgendaService } from
    '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
```

```

imports: [
    ScheduleModule,
    ButtonModule
],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, DragAndDropService, MonthAgendaService],
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStart) = "onDragStart($event)" ></ejs-
schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
    };
    onDragStart(args: DragEventArgs): void {
        args.interval = 10; // drag interval time is changed to 10 minutes
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Drag and drop items from external source

It is possible to drag and drop the unplanned items from any of the external source into the scheduler, by manually saving those dropped item as a new appointment data through **addEvent** method of Scheduler.

In this example, we have used the tree view control as an external source and the child nodes from the tree view component are dragged and dropped onto the Scheduler. Therefore, it is necessary to make use of the **nodeDragStop** event of the TreeView component, where we can form an event object and save it using the **addEvent** method.

APP.COMPONENT.HTML

```

<div class="control-section">
    <div class="drag-sample-wrapper">
        <div class="schedule-container">
            <div class="title-container">
                <h1 class="title-text">Doctor's Appointments</h1>
            </div>
            <ejs-schedule #scheduleObj cssClass='schedule-drag-drop'
width='100%' height='650px' [group]="group"
                [currentView]="currentView" [selectedDate]="selectedDate"
[workHours]="workHours" [eventSettings]="eventSettings"

```

```

                (actionBegin)="onActionBegin($event)"
            (drag)="onItemDrag($event)">
                <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'
                        [allowMultiple]='allowMultiple' textField='Text'
idField='Id' groupIDField='GroupID' colorField='Color'>
                    </e-resource>
                </e-resources>
                <ng-template #resourceHeaderTemplate let-data>
                    <div class="template-wrap">
                        <div class="specialist-category">
                            <div *ngIf="getConsultantStatus(data)">
                                
                                </div>
                                <div class="specialist-
name">{{getConsultantName(data)}}</div>
                                <div class="specialist-
designation">{{getConsultantDesignation(data)}}</div>
                            </div>
                        </div>
                    </ng-template>
                <e-views>
                    <e-view option='TimelineDay'></e-view>
                    <e-view option='TimelineMonth'></e-view>
                </e-views>
            </ejs-schedule>
        </div>
        <div class="treeview-container">
            <div class="title-container">
                <h1 class="title-text">Waiting List</h1>
            </div>
            <ejs-treeview #treeObj [fields]='field' cssClass='treeview-
external-drag' [allowDragAndDrop]='allowDragAndDrop'
                (nodeDragStop)="onTreeDragStop($event)"
            (nodeDragging)="onItemDrag($event)"></ejs-treeview>
        </div>
    </div>
</div>

```

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { CommonModule } from '@angular/common'
import { ScheduleAllModule, RecurrenceEditorAllModule } from
'@syncfusion/ej2-angular-schedule'
import { NumericTextBoxAllModule } from '@syncfusion/ej2-angular-inputs'
import { DatePickerAllModule, TimePickerAllModule, DateTimePickerAllModule }
from '@syncfusion/ej2-angular-calendars'

```

```

import { CheckBoxAllModule } from '@syncfusion/ej2-angular-buttons'
import { ToolbarAllModule, ContextMenuAllModule, TreeViewAllModule } from
 '@syncfusion/ej2-angular-navigations'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { DropDownListAllModule, MultiSelectAllModule } from '@syncfusion/ej2-
angular-dropdowns'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { hospitalData, waitingList } from './datasource';
import { extend, closest, remove, addClass } from '@syncfusion/ej2-base';
import {
    EventSettingsModel, View, GroupModel, TimelineViewsService,
    TimelineMonthService,
    ResizeService, WorkHoursModel, DragAndDropService, ResourceDetails,
    ScheduleComponent, ActionEventArgs, CellClickEventArgs
} from '@syncfusion/ej2-angular-schedule';
import { DragAndDropEventArgs } from '@syncfusion/ej2-navigations';
import { TreeViewComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
    imports: [CommonModule, ScheduleAllModule, RecurrenceEditorAllModule,
    NumericTextBoxAllModule,
        DatePickerAllModule, TimePickerAllModule, DateTimePickerAllModule,
    CheckBoxAllModule, ToolbarAllModule,
        DropDownListAllModule, ContextMenuAllModule, TreeViewAllModule,
        MaskedTextBoxModule, MultiSelectAllModule, ],
    standalone: true,
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./index.css'],
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    @ViewChild('treeObj')
    public treeObj?: TreeViewComponent;
    public isTreeItemDropped: boolean = false;
    public draggedItemId: string = '';
    public selectedDate: Date = new Date(2018, 7, 1);
    public currentView: View = 'TimelineDay';
    public workHours: WorkHoursModel = { start: '08:00', end: '18:00' };
    public departmentDataSource: Object[] = [
        { Text: 'GENERAL', Id: 1, Color: '#bbdc00' },
        { Text: 'DENTAL', Id: 2, Color: '#9e5fff' }
    ];
    public consultantDataSource: Object[] = [
        { 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' }
    ]
}

```



```

];
    public group: GroupModel = { enableCompactView: false, resources:
['Departments', 'Consultants'] };
    public allowMultiple: Boolean = false;
    public eventSettings: EventSettingsModel = {
        dataSource: hospitalData,
        fields: {
            subject: { title: 'Patient Name', name: 'Name' },
            startTime: { title: 'From', name: 'StartTime' },
            endTime: { title: 'To', name: 'EndTime' },
            description: { title: 'Reason', name: 'Description' }
        }
    };
    public field: Object = { dataSource: waitingList, id: 'Id', text: 'Name'
};
    public allowDragAndDrop: boolean = true;
    getConsultantName(value: ResourceDetails): string {
        return (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as any).textField] as string;
    }
    getConsultantStatus(value: ResourceDetails): boolean {
        let resourceName: string =
            (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as any).textField] as string;
        if (resourceName === 'GENERAL' || resourceName === 'DENTAL') {
            return false;
        } else {
            return true;
        }
    }
    getConsultantDesignation(value: ResourceDetails): string {
        let resourceName: string =
            (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as any).textField] as string;
        if (resourceName === 'GENERAL' || resourceName === 'DENTAL') {
            return '';
        } else {
            return (value as ResourceDetails).resourceData['Designation'] as
string;
        }
    }
    getConsultantImageName(value: ResourceDetails): string {
        return this.getConsultantName(value).toLowerCase();
    }
    onItemDrag(event: any): void {
        if (this.scheduleObj?.isAdaptive) {
            let classElement: HTMLElement =
this.scheduleObj.element.querySelector('.e-device-hover') as HTMLElement;
            if (classElement) {
                classElement.classList.remove('e-device-hover');
            }
            if (event.target.classList.contains('e-work-cells')) {
                addClass([event.target], 'e-device-hover');
            }
        }
        if (document.body.style.cursor === 'not-allowed') {
            document.body.style.cursor = '';
        }
    }

```

```

    }
    if (event.name === 'nodeDragging') {
        let dragElementIcon: NodeListOf<HTMLElement> =
            document.querySelectorAll('.e-drag-item.treeview-external-
drag .e-icon-expandable');
        for (let i: number = 0; i < dragElementIcon.length; i++) {
            dragElementIcon[i].style.display = 'none';
        }
    }
}
onActionBegin(event: ActionEventArgs): void {
    if (event.requestType === 'eventCreate' && this.isTreeItemDropped) {
        let treeViewdata: { [key: string]: Object }[] =
this.treeObj?.fields.dataSource as { [key: string]: Object }[];
        const filteredPeople: { [key: string]: Object }[] =
            treeViewdata.filter((item: any) => item.Id !==
parseInt(this.draggedItemId, 10));
        this.treeObj!.fields.dataSource = filteredPeople;
        let elements: NodeListOf<HTMLElement> =
document.querySelectorAll('.e-drag-item.treeview-external-drag');
        for (let i: number = 0; i < elements.length; i++) {
            remove(elements[i]);
        }
    }
}
onTreeDragStop(event: DragAndDropEventArgs): void {
    let treeElement: Element = <Element>closest(event.target, '.e-
treeview');
    let classElement: HTMLElement =
this.scheduleObj?.element.querySelector('.e-device-hover') as HTMLElement;
    if (classElement) {
        classElement.classList.remove('e-device-hover');
    }
    if (!treeElement) {
        event.cancel = true;
        let scheduleElement: Element = <Element>closest(event.target,
'.e-content-wrap');
        if (scheduleElement) {
            let treeviewData: { [key: string]: Object }[] =
                this.treeObj?.fields.dataSource as { [key: string]:
Object }[];
            if (event.target.classList.contains('e-work-cells')) {
                const filteredData: { [key: string]: Object }[] =
                    treeviewData.filter((item: any) => item.Id ===
parseInt(event.draggedNodeData['id'] as string, 10));
                let cellData: CellClickEventArgs =
this.scheduleObj!.getCellDetails(event.target);
                let resourceDetails: ResourceDetails =
this.scheduleObj!.getResourcesByIndex((cellData as any).groupIndex);
                let eventData: { [key: string]: Object } = {
                    Name: filteredData[0]['Name'],
                    StartTime: cellData.startTime,
                    EndTime: cellData.endTime,
                    IsAllDay: cellData.isAllDay,
                    Description: filteredData[0]['Description'],
                    DepartmentID:
resourceDetails.resourceData['GroupId'],

```

```

        ConsultantID: resourceDetails.resourceData['Id']
    };
    this.scheduleObj?.addEvent(eventData);
    this.isTreeItemDropped = true;
    this.draggedItemId = event.draggedNodeData['id'] as
string;
    }
    }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Opening the editor window on drag stop

There are scenarios where you want to open the editor filled with data on newly dropped location and may need to proceed to save it, only when **Save** button is clicked on the editor. On clicking the cancel button should revert these changes. This can be achieved using the **dragStop** event of Scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { scheduleData } from './datasource';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, DragAndDropService, DragEventArgs,
    ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        DragAndDropService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component

```

```

    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings" (dragStop)="onTreeDragStop($event)"></ejs-
schedule>`
  })
  export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: scheduleData,
    };
    onTreeDragStop(event: DragEventArgs): void {
      event.cancel = true; // cancels the drop action
      this.scheduleObj?.openEditor(event.data, 'Save'); // open the event
window with updated start and end time
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Inline Appointment

In Scheduler, another easier way for adding or editing the appointment's subject alone can be achieved by using the inline Add/Edit support. It allows the user to add and edit the appointments inline. To get familiar with the inline Add mode, single click on any of the Scheduler cells or press enter key on the selected cells.

When the inline adding mode is ON, a text box will get created within the clicked Scheduler cells with a blinking cursor in it, requiring the user to enter the subject of an appointment. Once the subject is entered, the appointment will be saved on pressing the enter key.

To enable the inline edit mode, single click on any of the existing appointment's subject, so that the user can edit the subject of that appointment. The edited subject of that appointment will be updated on pressing the enter key.

The inline option can be enabled/disabled on the Scheduler by using the allowInline API, whereas its default value is set to false.

While using the allowInline the showQuickInfo will be turned off. The quickPopup will not show on clicking the work cell or clicking the appointment when the allowInline property is set to true.

In work cells, select multiple cells using keyboard, and then press enter key. The appointment wrapper will be created, and focus will be on the subject field. Also, consider the overlapping scenarios when creating an inline event.

Normal Event

While editing appointments, single-click the appointment subject, the editable option will be enabled in UI and the cursor will focus at the end of the text. Inline editing will be considered for all possible views.

Recurrence Event

While editing the occurrence from the recurrence series, it is only possible to edit a **single occurrence**, not an entire series.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, View, EventSettingsModel, MonthAgendaService } from
'@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [currentView]='currentView'
[eventSettings]='eventSettings' [allowInline]="allowInline"
></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2019, 0, 17);
  public allowInline: boolean = true;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public currentView: View = 'Month';
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Appointment Resizing

Another way of rescheduling an appointment can be done by resizing it through either of its handlers. To work with resizing functionality, it is necessary to inject the module **Resize** and make sure that **allowResizing** property is set to true.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
```

```

import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, ResizeService
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        ResizeService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Disable the resize action

By default, resizing of events is allowed on all Scheduler views except Agenda and Month-Agenda view. To disable this event resizing action, set false to the `allowResizing` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import {

```

```

    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, ResizeService
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ResizeService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[allowResizing] = 'false' [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Disable scrolling on resize action

By default, while resizing an appointment, when its handler reaches the extreme edges of the Scheduler, scrolling action will take place along with event resizing. To prevent this scrolling action, set false to `scroll` value within the `resizeStart` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import {
  EventSettingsModel, DayService, WeekService, WorkWeekService,
  MonthService, AgendaService, ResizeService, ResizeEventArgs
} from '@syncfusion/ej2-angular-schedule';

```

```

@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ResizeService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(resizeStart)="onResizeStart($event)"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onResizeStart(event: ResizeEventArgs): void {
    event.scroll = { enable: false,
      scrollBy: 30,
      timeDelay: 500
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Controlling scroll speed while resizing an event

The speed of the scrolling action while resizing an appointment to the Scheduler edges, can be controlled within the `resizeStart` event by setting the desired value to the `scrollBy` option.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import {

```



```

    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, ResizeService, ResizeEventArgs
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ResizeService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(resizeStart)="onResizeStart($event)"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onResizeStart(event: ResizeEventArgs): void {
    event.scroll = { enable: true, scrollBy: 15, timeDelay: 500 };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Setting resize time interval

By default, while resizing an appointment, it extends or shrinks at an interval of 30 minutes. To change this default resize interval, set appropriate values to `interval` option within the `resizeStart` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';

```

```

import {
  EventSettingsModel, ResizeService, ResizeEventArgs
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ResizeService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(resizeStart)="onResizeStart($event)"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
  onResizeStart(event: ResizeEventArgs): void {
    event.interval = 10; //resize interval time is changed to 10 minutes
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Appointment customization

The look and feel of the Scheduler events can be customized using any one of the following ways.

- [Using event template](#)
- [Using eventRendered event](#)
- [Using customCSS class](#)

Using template

Any kind of text, images and links can be added to customize the look of the events. The user can format and change the default appearance of the events by making use of the `template` option available within the [Link to the Video](#) property. The following code example customizes the appointment's default color and time format.

Learn how easily you can customize the basic look and feel of Angular Scheduler appointments using its built-in ng-template option from this video:

APP.COMPONENT.HTML

```
<ejs-schedule width='100%' height='550px' [selectedDate]="selectedDate"
cssClass='schedule-event-template' [readonly]='readonly'
[eventSettings]="eventSettings">
  <ng-template #eventSettingsTemplate let-data>
    <div class='template-wrap' [style.background]="data.SecondaryColor">
      <div class="subject"
[style.background]="data.PrimaryColor">{{data.Subject}}</div>
      <div class="time" [style.background]="data.PrimaryColor">Time:
{{getTimeString(data.StartTime)}} - {{getTimeString(data.EndTime)}}</div>
    </div>
  </ng-template>
</ejs-schedule>
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { webinarData } from './datasource';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  templateUrl: './app.component.html',
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: webinarData,
  };
  private instance: Internationalization = new Internationalization();
  readonly: any;
```

```

getTimeString(value: Date): string {
    return this.instance.formatDate(value, { skeleton: 'hm' });
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

All the built-in fields that are mapped to the appropriate field properties within the `eventSettings`, as well as custom mapped fields from the Scheduler dataSource can be accessed within the template code.

Using `eventRendered` event

The `eventRendered` event triggers before the appointment renders on the Scheduler. Therefore, this client-side event can be utilized to customize the look of events based on any specific criteria, before rendering them on the scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { scheduleData } from './datasource';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, View, EventRenderedArgs, ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(eventRendered)="applyCategoryColor($event)"></ejs-schedule>`
})
export class AppComponent {
    @ViewChild('scheduleObj')

```

```

public scheduleObj?: ScheduleComponent;
public selectedDate: Date = new Date(2018, 1, 15);
public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
};
applyCategoryColor(args: EventRenderedArgs): void {
    let categoryColor: string = args.data['CategoryColor'] as string;
    if (!args.element || !categoryColor) {
        return;
    }
    if (this.scheduleObj?.currentView === 'Agenda') {
        (args.element.firstChild as HTMLElement).style.borderLeftColor =
categoryColor;
    } else {
        args.element.style.backgroundColor = categoryColor;
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Using cssClass

The customization of events can also be achieved using `cssClass` property of the Scheduler. In the following example, the background of appointments has been changed using the `cssClass`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { scheduleData } from './datasource';
import {
    EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
})

```

```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj cssClass='custom-
class' height='550px' [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`,
  styles: [`.custom-class.e-schedule .e-vertical-view .e-all-day-
appointment-wrapper .e-appointment,
.custom-class.e-schedule .e-vertical-view .e-day-wrapper .e-appointment,
.custom-class.e-schedule .e-month-view .e-appointment{
  background: #006400;
} `],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Setting minimum height

It is possible to set minimal height for appointments on Scheduler using `eventRendered` event, when its start and end time duration is less than the default duration of a single slot.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import {
  EventSettingsModel, EventRenderedArgs, ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,

```

```

        MonthAgendaService],
standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(eventRendered)="setMinimumHeight($event)"></ejs-schedule>`
  })
  export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public data = [{
      Id: 13,
      Subject: 'Myths of Andromeda Galaxy',
      StartTime: new Date(2018, 1, 16, 10, 30),
      EndTime: new Date(2018, 1, 16, 10, 40)
    }, {
      Id: 14,
      Subject: 'Aliens vs Humans',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 15, 10, 20)
    }];
    public eventSettings: EventSettingsModel = {
      dataSource: this.data,
    };
    setMinimumHeight(args: EventRenderedArgs): void {
      if (this.scheduleObj?.currentView !== 'Month') {
        let cellHeight =
          (this.scheduleObj?.element.querySelector('.e-work-cells') as
            HTMLElement).offsetHeight;
        let appHeight: number = (args.data['EndTime'].getTime() -
          args.data['StartTime'].getTime())
          / (60 * 1000) * (cellHeight * (this.scheduleObj as
            any).timeScale.slotCount) / (this.scheduleObj as any).timeScale.interval;
        args.element.style.height = appHeight + 'px';
      }
    }
  }
}

```

MAIN.TS

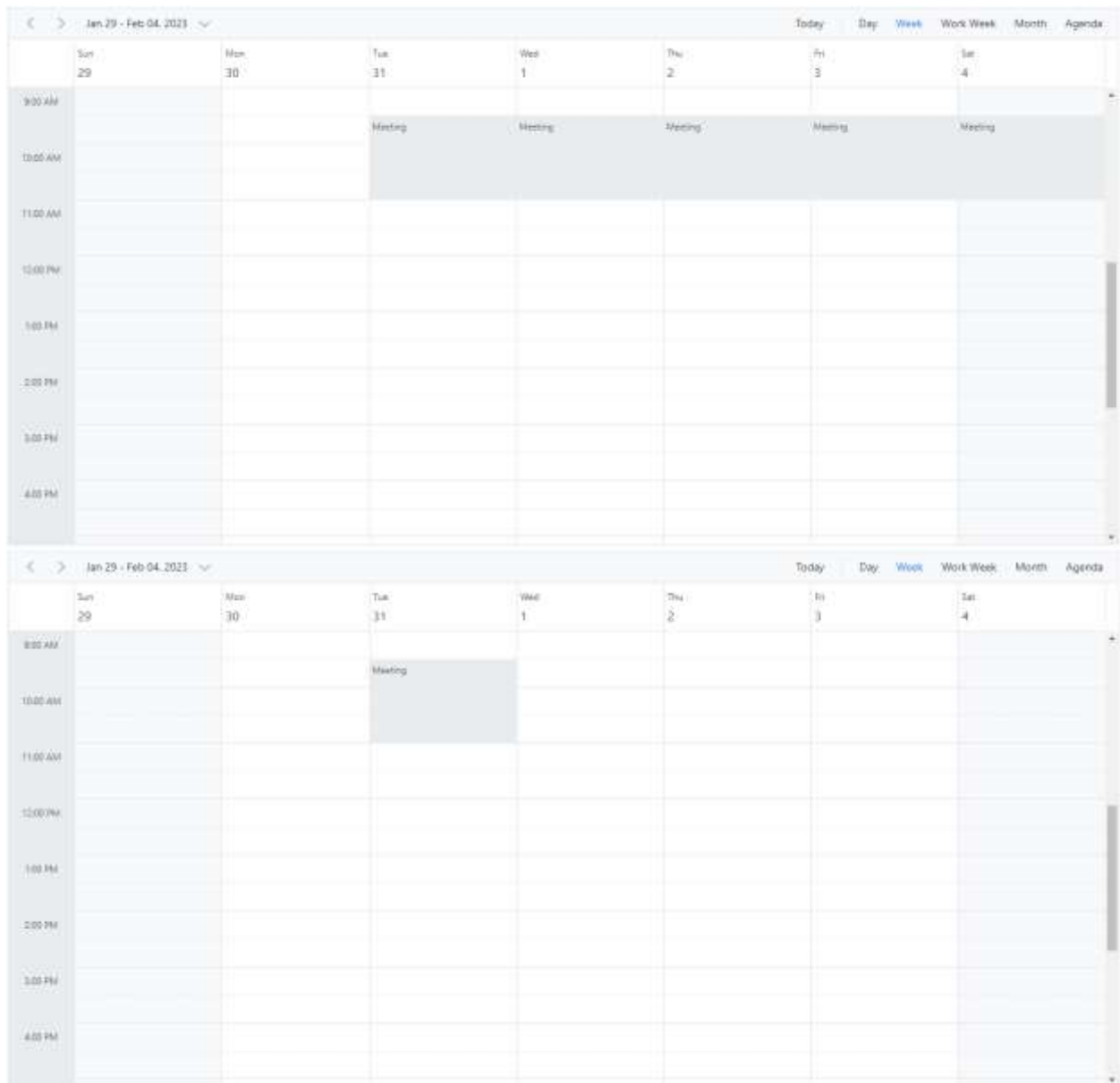
```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Block Dates and Times

It is possible to block a set of dates or a particular time ranges on the Scheduler. To do so, define an appointment object within `eventSettings` along with the required time range to block and set the `isBlock` field to true. Usually, the event objects defined with `isBlock` field set to true will block the entire time cells lying within the appropriate time ranges specified through `startTime` and



endTime fields.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { blockData } from './datasource';
import {
    EventSettingsModel, TimelineViewsService,
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
```



```

        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService,
                TimelineViewsService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: blockData,
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Block Date and Time](images/schedule-block-events.png)

Block events can also be defined to repeat on several days as shown in the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import {
    EventSettingsModel, TimelineViewsService
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
              WeekService,

```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public data: object [] = [{
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2018, 1, 15, 9, 30),
        EndTime: new Date(2018, 1, 15, 11, 0),
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=5',
        IsBlock: true
    }, {
        Id: 2,
        Subject: 'Thule Air Crash Report',
        StartTime: new Date(2018, 1, 14, 12, 0),
        EndTime: new Date(2018, 1, 14, 14, 0)
    }];
    public eventSettings: EventSettingsModel = {
        dataSource: this.data,
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Block Several Date and Time](images/schedule-multiple-block-events.png)

Readonly

An interaction with the appointments of Scheduler can be enabled/disabled using the **readonly** property. With this property enabled, you can simply navigate between the Scheduler dates, views and can be able to view the appointment details in the quick info window. Most importantly, the users are not allowed to perform any CRUD actions on Scheduler, when this property is set to true. By default, it is set as **false**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'

```

```
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import {
    EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [

        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[readonly]="true" [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
    };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Make specific events readonly

There are scenarios where you need to restrict the CRUD action on specific appointments alone based on certain conditions. In the following example, the events that has occurred on the past hours from the current date of the Scheduler are made as read-only and the CRUD actions has been prevented only on those appointments. This can be achieved by setting `isReadonly` field of read-only events to `true`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
```

```
import { readOnlyData } from './datasource';
import {
    EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [

        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: readOnlyData,
    };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

By default, the event editor is prevented to open on the read-only events when `isReadOnly` field is set to `true`.

Restricting event creation on specific time slots

You can restrict the users to create and update more than one appointment on specific time slots. Also, you can disable the CRUD action on those time slots if it is already occupied, which can be achieved using Scheduler's public method `isSlotAvailable`.

Note: The `isSlotAvailable` is centered around verifying appointments within the present view's date range. Yet, it does not encompass an evaluation of availability for recurrence occurrences that fall beyond this particular date range.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
```

```

import { WeekService, AgendaService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { scheduleData } from './datasource';
import {
    DayService, TimelineViewsService, WorkWeekService, MonthService,
    ActionEventArgs, ScheduleComponent,
    EventSettingsModel, EventFieldsMapping
} from '@syncfusion/ej2-angular-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({
    imports: [

        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(actionBegin)="onActionBegin($event)"></ejs-schedule>`
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
    };
    onActionBegin(args: ActionEventArgs) {
        if ((args.requestType === 'eventCreate' || args.requestType ===
'eventChange') && ((<Object[]>args.data).length > 0
|| !isNullOrUndefined((args as any).data)) {
            let eventData: any = args.data as any;
            let eventField: EventFieldsMapping | undefined =
this.scheduleObj?.eventFields;
            let startDate: Date = (((<Object[]>args.data).length > 0)
? eventData[0][(eventField as any).startTime] :
eventData[(eventField as any).startTime]) as Date;
            let endDate: Date = (((<Object[]>args.data).length > 0)
? eventData[0][(eventField as any).endTime] :
eventData[(eventField as any).endTime]) as Date;
            args.cancel = !this.scheduleObj?.isSlotAvailable(startDate,
endDate);
        }
    }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Differentiate the past time events

To differentiate the appearance of the appointments based on specific criteria such as displaying the past hour appointments with different colors on Scheduler `eventRendered` event can be used which triggers before the appointment renders on the Scheduler.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { WeekService, AgendaService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { scheduleData } from './datasource';
import {
    DayService, TimelineViewsService, WorkWeekService, MonthService,
    EventRenderedArgs, ScheduleComponent, EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
cssClass='custom-class' [selectedDate]="selectedDate"
[eventSettings]="eventSettings"
(eventRendered)="onEventRendered($event)"></ejs-schedule>`,
    styleUrls: ['./index.css'],
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
```

```

        dataSource: scheduleData,
    };
    public isReadOnly: Function = (data: { [key: string]: Object }): boolean
=> {
        return (data['EndTime'] < (this.scheduleObj as any).selectedDate);
    };
    onEventRendered(args: EventRenderedArgs) {
        if (this.isReadOnly(args.data)) {
            args.element.classList.add('e-past-app');
        }
    }
}

```

MAIN.TS

```

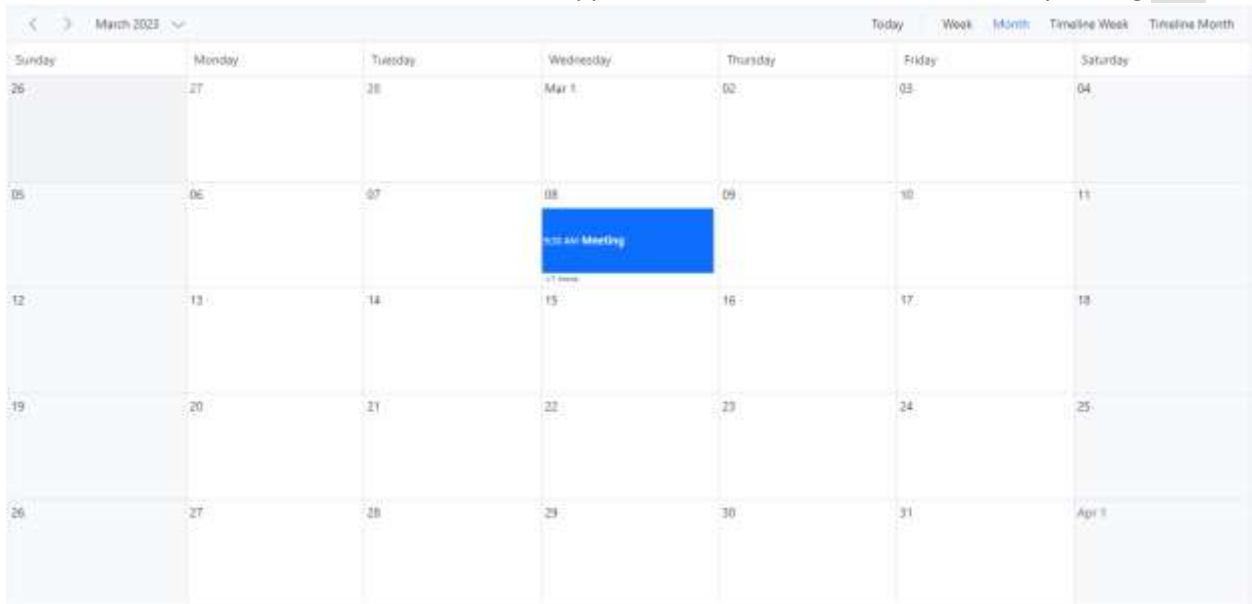
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Appointments occupying entire cell

The Scheduler allows the event to occupies the full height of the cell without its header part by setting **true** for **enableMaxHeight** Property.

We can show more indicator if more than one appointment is available in a same cell by setting **true** to



enableIndicator property whereas its default value is false.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';

```

```

import { Component, ViewChild } from '@angular/core';
import { scheduleData } from './datasource';
import {
    TimelineViewsService, TimelineMonthService, EventSettingsModel,
    ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService, TimelineMonthService]],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' #scheduleObj height='550px'
currentView="TimelineMonth" [selectedDate]="selectedDate"
[eventSettings]="eventSettings">
    <e-views>
        <e-view option='TimelineWeek'></e-view>
        <e-view option='TimelineMonth'></e-view>
    </e-views>
</ejs-schedule>`,
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
        enableMaxHeight: true,
        enableIndicator: false
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Differentiate Past Time Events](images/schedule-appointments-entirecell.png)

How to limit maximum number of events to display

In the Scheduler, the default behavior is to display concurrent events based on cell height, with each new event represented as

+n more characters. However, you may want to improve the quality of the presentation by limiting the number of concurrent events. This can be accomplished by using the [maxEventsPerRow](#) property, which is defaulted to the [views](#) property.

The [maxEventsPerRow](#) property is specific to the month, timeline month, and timeline year views, allowing you to view events visually in these rows. Below is a code example that demonstrates how to use this constraint and the events displayed in a cell have been created:

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthService, TimelineMonthService, TimelineYearService } from
 '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import {
    EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
    imports: [

        ScheduleModule,
        TimePickerModule
    ],
    providers: [TimelineMonthService, MonthService, TimelineYearService],
    standalone: true,
    selector: "app-root",
    // specifies the template string for the Schedule component
    template: `
        <ejs-schedule width="100%" height="380px" [selectedDate]="selectedDate"
        [eventSettings]="eventSettings">
            <e-views>
                <e-view option="Month" [maxEventsPerRow]="3"></e-view>
            </e-views>
        </ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2023, 11, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
    };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The property [maxEventsPerRow](#) will be applicable only when [rowAutoHeight](#) feature is disabled in the Scheduler.

Display tooltip for appointments

The tooltip shows the Scheduler appointment's information in a formatted style by making use of the tooltip related options.

Show or hide built-in tooltip

The tooltip can be displayed for appointments by setting `true` to the `enableTooltip` option within the `eventSettings` property.

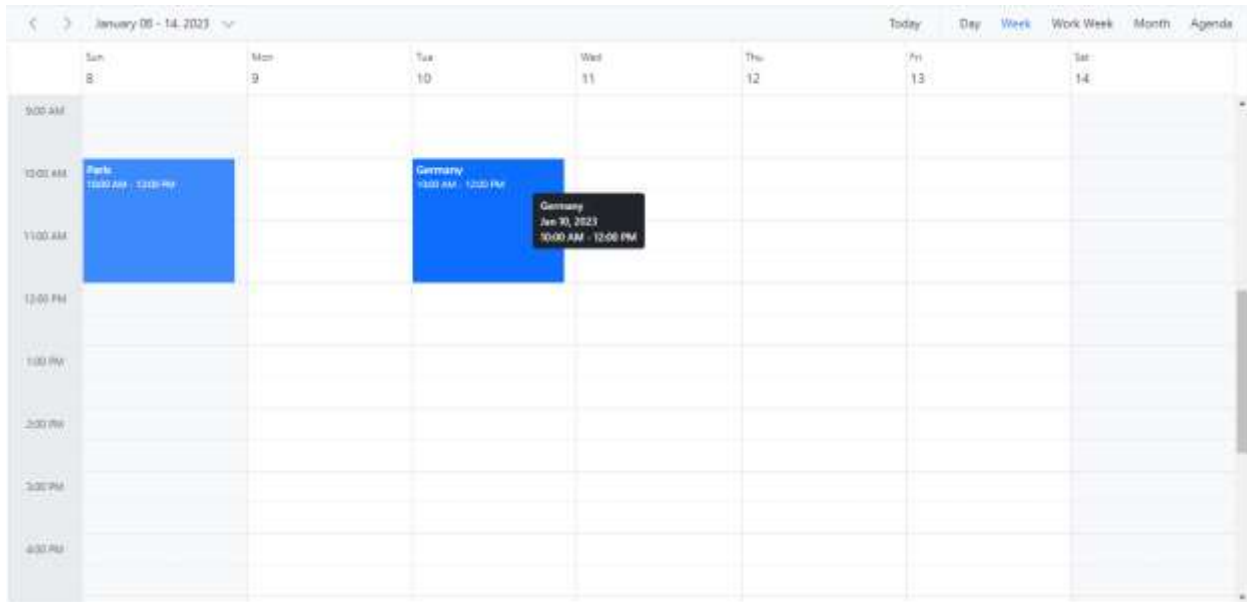
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { scheduleData } from '../datasource';
import { TimelineViewsService, EventSettingsModel, ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`,
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
    enableTooltip: true
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Customizing event tooltip using template

After enabling the default tooltip, it is possible to customize the display of needed event information on tooltip by making use of the `tooltipTemplate` option within the `eventSettings`.

APP.COMPONENT.HTML

```
<ejs-schedule width='100%' height='550px' [selectedDate]="selectedDate"
cssClass='e-schedule-event-tooltip' [(currentView)]="currentView"
[eventSettings]="eventSettings">
  <ng-template #eventSettingsTooltipTemplate let-data>
    <div class="tooltip-wrap">
      <div class="image {{data.EventType}}"></div>
      <div class="content-area"><div class="name">{{data.Subject}}</div>

      <div *ngIf="data.City != null && data.City !=undefined">
        <div class="city">{{data.City}}</div>
      </div>
      <div class="time">From&#160;:&#160;{{data.StartTime.toLocaleString()}}

</div>
      <div
class="time">To&#160;:&#160;{{data.EndTime.toLocaleString()}} </div>
      </div></div>

    </ng-template>
  </ejs-schedule>
```

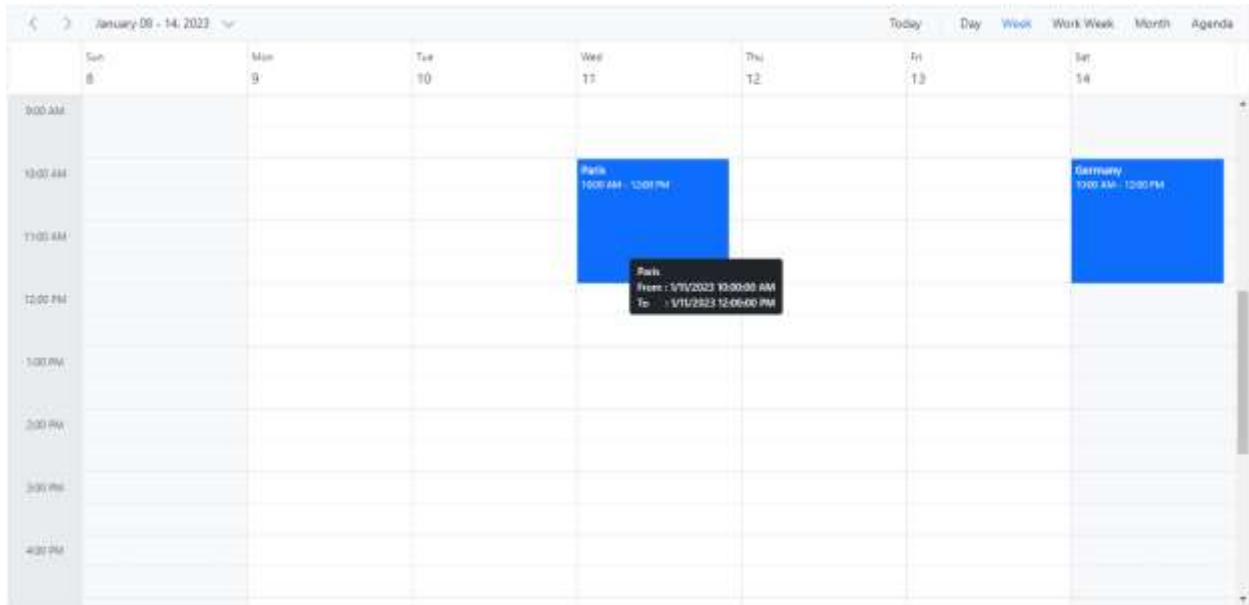
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
```

```
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService, EventSettingsModel}
from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { eventsData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  templateUrl: './app.component.html',
  styleUrls: ['./index.css'],
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: eventsData,
    enableTooltip: true,
  };
  currentView: any;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



All the field names that are mapped from the Scheduler dataSource to the appropriate field properties such as subject, description, location, startTime and endTime within the `eventSettings` can be accessed within the template.

Appointment filtering

The appointments can be filtered by passing the predicate value to `query` option in `eventSettings`. The following code example shows how to filter and render the selected appointments alone in the Scheduler.

APP.COMPONENT.HTML

```
<table id="property" title="Filter events" style="width: 40%">
  <tbody>
    <tr>
      <td>
        <ejs-checkbox #confirmObj id="confirmed" label='Confirmed'
        [checked]="isChecked" (change)="onCheckBoxChange()"></ejs-checkbox>
      </td>
      <td>
        <ejs-checkbox #requestedObj id="requested" label='Requested'
        [checked]="isChecked" (change)="onCheckBoxChange()"></ejs-checkbox>
      </td>
      <td>
        <ejs-checkbox #freshObj id="new" label='New'
        [checked]="isChecked" (change)="onCheckBoxChange()"></ejs-checkbox>
      </td>
    </tr>
  </tbody>
</table>
<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(eventRendered)="onEventRendered($event)"></ejs-schedule>
```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { CheckboxModule } from '@syncfusion/ej2-angular-buttons'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { Checkbox } from '@syncfusion/ej2-buttons';
import { Query, Predicate } from '@syncfusion/ej2-data';
import {
    ScheduleComponent, MonthService, DayService, WeekService, WorkWeekService,
    EventSettingsModel, AgendaService, EventRenderedArgs
} from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from '../datasource';
@Component({
    imports: [
        CheckboxModule,
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    templateUrl: './app.component.html'
})
export class AppComponent {
    @ViewChild('confirmObj')
    public confirmObj?: Checkbox;
    @ViewChild('requestedObj')
    public requestedObj?: Checkbox;
    @ViewChild('freshObj')
    public freshObj?: Checkbox;
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public selectedDate: Date = new Date(2018, 1, 15);
    public isChecked: Boolean = true;
    public onEventRendered(args: EventRenderedArgs): void {
        switch (args.data['EventType']) {
            case 'Requested':
                (args.element as HTMLElement).style.backgroundColor =
                    '#F57F17';
                break;
            case 'Confirmed':
                (args.element as HTMLElement).style.backgroundColor =
                    '#7fa900';
                break;
            case 'New':
                (args.element as HTMLElement).style.backgroundColor =
                    '#8e24aa';

```

```

        break;
    }
}
public onCheckBoxChange(): void {
    let predicate: Predicate | undefined;
    const checkBoxes: CheckBox[] = [this.confirmObj as any,
this.requestedObj, this.freshObj];
    checkBoxes.forEach((checkBoxObj: CheckBox) => {
        if (checkBoxObj.checked) {
            if (predicate) {
                predicate = predicate.or('EventType', 'equal',
checkBoxObj.label);
            } else {
                predicate = new Predicate('EventType', 'equal',
checkBoxObj.label);
            }
        }
    });
    (this.scheduleObj as any).eventSettings.query = new
Query().where(predicate || '');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Appointment selection

Appointment selection can be done either through mouse or keyboard actions. The selected events in UI will have a box shadow effect around to differentiate it from other appointments.

Action	Description
----- -----	
Mouse click or Single tap on appointments	Selects single appointment.
Ctrl + [Mouse click] or [Single tap] on appointments	Selects multiple appointments.

Deleting multiple appointments

With the options available to select multiple appointments, it is also possible to delete the multiple selected appointments simply by pressing the **delete** key. In case of deleting multiple selected occurrences of an event series, only those occurrences will be deleted and not the entire series.

Retrieve event details from the UI of an event

It is possible to access the information about the event fields of an appointment element displayed on the Scheduler UI. This can be achieved by passing an appointment element as argument to the public method **getEventDetails**.

In the following example, the subject of the appointment clicked has been displayed.

APP.COMPONENT.HTML

```

<div id='container' class="col-lg-12">
  <div class="content-wrapper">
    <div class="col-lg-9 control-section">
      <ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(eventClick)="onEventClick($event)" ">
    </ejs-schedule>
  </div>
  <div class="col-lg-3 property-section">
    <table id="property" title="Event Trace">
      <tbody>
        <tr>
          <td>
            <div class="eventarea" style="height:
245px;overflow: auto">
              <span class="EventLog" id="EventLog"
style="word-break: normal;" #eventLog>
                <span *ngFor="let log of eventLogs">{{
log }} <hr></span>
              </span>
            </div>
          </td>
        </tr>
        <tr>
          <td>
            <div class="evtbtn" style="padding-bottom: 10px">
              <input id="clear" type="button"
(click)="onClick()" value="Clear">
            </div>
          </td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
</div>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ElementRef } from '@angular/core';
import { eventsData } from '../datasource';
import {
  TimelineViewsService, EventClickArgs, EventSettingsModel,
ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule

```



```

    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService,
                TimelineViewsService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    templateUrl: './app.component.html',
    styleUrls: ['./index.css'],
  })
  export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    @ViewChild('eventLog') eventLog!: ElementRef<HTMLSpanElement>;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: eventsData
    };
    public eventLogs: string[] = [];
    onClick() {
      this.eventLogs = [];
    }
    onEventClick(args: EventClickArgs): void {
      let event: Object = (this.scheduleObj as
any).getEventDetails(args.element);
      let innerHtml = (event as any).Subject;
      this.eventLogs.push(`${innerHtml}`);
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Get the current view appointments

To retrieve the appointments present in the current view of the Scheduler, you can make use of the `getCurrentViewEvents` public method. In the following example, the count of current view appointment collection rendered has been traced in `dataBound` event.

APP.COMPONENT.HTML

```

<div id='container' class="col-lg-12">
  <div class="content-wrapper">
    <div class="col-lg-9 control-section">
      <ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(dataBound)="onDataBound()" ">
    </ejs-schedule>

```

```

        </div>
        <div class="col-lg-3 property-section">
            <table id="property" title="Event Trace">
                <tbody>
                    <tr>
                        <td>
                            <div class="eventarea" style="height:
245px;overflow: auto">
                                <span class="EventLog" id="EventLog"
style="word-break: normal;">
                                    <span *ngFor="let log of eventLogs">{{
log }} <hr></span>
                                </span>
                            </div>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <div class="evtbtn" style="padding-bottom: 10px">
                                <input id="clear" type="button"
(click)="onClick()" value="Clear">
                            </div>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
    </div>
</div>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { eventsData } from '../datasource';
import { EventSettingsModel,
    ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,

```

```

    selector: 'app-root',
    // specifies the template string for the Schedule component
    templateUrl: './app.component.html',
    styleUrls: ['./index.css'],
  })
  export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: eventsData
    };
    public eventLogs: string[] = [];
    onClick() {
      this.eventLogs = [];
    }
    onDataBound(): void {
      let event: Object[] = (this.scheduleObj as
any).getCurrentViewEvents();
      if (event.length > 0) {
        this.appendElement(`Events present on current view
${event.length}`);
      } else {
        this.appendElement('No Events available in this view.');
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Get the entire appointment collections

The entire collection of appointments rendered on the Scheduler can be accessed using the `getEvents` public method. In the following example, the count of entire appointment collection rendered on the Scheduler has been traced in `dataBound` event.

APP.COMPONENT.HTML

```

<div id='container' class="col-lg-12">
  <div class="content-wrapper">
    <div class="col-lg-9 control-section">
      <ejs-schedule width='100%' #scheduleObj height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(dataBound)="onDataBound()">
        </ejs-schedule>
      </div>
      <div class="col-lg-3 property-section">
        <table id="property" title="Event Trace">

```

```

        <tbody>
          <tr>
            <td>
              <div class="eventarea" style="height:
245px;overflow: auto">
                <span class="EventLog" id="EventLog"
style="word-break: normal;">
                  <span *ngFor="let log of
eventLogs">Events present on scheduler <b>{{ log }}</b> <hr></span>
                </span>
              </div>
            </td>
          </tr>
          <tr>
            <td>
              <div class="evtbtn" style="padding-bottom: 10px">
                <input id="clear" type="button"
(click)="onClick()" value="Clear">
              </div>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</div>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { eventsData } from './datasource';
import { EventSettingsModel,
ScheduleComponent
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  templateUrl: './app.component.html',

```

```

        styleUrls: ['./index.css'],
    })
    export class AppComponent {
        @ViewChild('scheduleObj')
        public scheduleObj?: ScheduleComponent;
        public selectedDate: Date = new Date(2018, 1, 15);
        public eventSettings: EventSettingsModel = {
            dataSource: eventsData
        };
        public eventLogs: string[] = [];
        onClick() {
            this.eventLogs = [];
        }
        onDataBound(): void {
            let event: Object[] = (this.scheduleObj as any).getEvents();
            this.appendElement(' ' + event.length + ' ');
        }
        appendElement(html: string): void {
            this.eventLogs.push(`${html}`);
        }
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Refresh appointments

If your requirement is to simply refresh the appointments instead of refreshing the entire Scheduler elements from your application end, make use of the `refreshEvents` public method.

```
`typescript
```

```
this.scheduleObj.refreshEvents();
```

```
,
```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Data binding in Angular Schedule component

The Scheduler uses `DataManager`, which supports both RESTful data service binding and JavaScript object array binding. The `dataSource` property of Scheduler can be assigned either with the instance of `DataManager` or JavaScript object array collection, as it supports the following two kind of data binding methods:

- Local data
- Remote data

Binding local data

To bind local JSON data to the Scheduler, you can simply assign a JavaScript object array to the [dataSource](#) option of the scheduler within the `eventSettings` property. The JSON object `dataSource` can also be provided as an instance of `DataManager` and assigned to the Scheduler `dataSource` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [eventSettings]='eventSettings'></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: [
      {
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2018, 1, 15, 9, 30),
        EndTime: new Date(2018, 1, 15, 11, 0)
      }, {
        Id: 2,
        Subject: 'Thule Air Crash Report',
        StartTime: new Date(2018, 1, 12, 12, 0),
        EndTime: new Date(2018, 1, 12, 14, 0)
      }, {
        Id: 3,
        Subject: 'Blue Moon Eclipse',
        StartTime: new Date(2018, 1, 13, 9, 30),
        EndTime: new Date(2018, 1, 13, 11, 0)
      }, {
        Id: 4,
        Subject: 'Meteor Showers in 2018',
        StartTime: new Date(2018, 1, 14, 13, 0),
        EndTime: new Date(2018, 1, 14, 14, 30)
      }
    ]
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

By default, **DataManager** uses **JsonAdaptor** for binding local data.

You can also bind different field names to the default event fields as well as include additional custom fields to the event object collection which can be referred [here](#).

Binding remote data

Any kind of remote data services can be bound to the Scheduler. To do so, create an instance of **DataManager** and provide the service URL to the **url** option of **DataManager** and then assign it to the **dataSource** property within **eventSettings**.

Using ODataV4Adaptor

ODataV4 is a standardized protocol for creating and consuming data. Refer to the following code example to retrieve the data from ODataV4 service using the **DataManager**. To connect with ODataV4 service end points, it is necessary to make use of **ODataV4Adaptor** within **DataManager**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService ],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public readonly: boolean = true;
  public selectedDate: Date = new Date(2020, 9, 20);
  private dataManager: DataManager = new DataManager({
    url: 'https://ej2services.syncfusion.com/production/web-
services/api/Schedule',
```

```

        adaptor: new ODataV4Adaptor,
        crossDomain: true
    });
    public eventSettings: EventSettingsModel = { dataSource: this.dataManager
};
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Filter events using the in-built query

To enable server-side filtering operations based on predetermined conditions, the [includeFiltersInQuery](#) API can be set to true, this allows the filter query to be constructed using the start date, end date, and recurrence rule which in turn enables the request to be filtered accordingly.

This method greatly improves the component's performance by reducing the data that needs to be transferred to the client side. As a result, the component's efficiency and responsiveness are significantly enhanced, resulting in a better user experience. However, it is important to consider the possibility of longer query strings, which may cause issues with the maximum URL length or server limitations on query string length.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public readonly: boolean = true;
  public selectedDate: Date = new Date(1996, 6, 9);
  private dataManager: DataManager = new DataManager({

```



```

url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
adaptor: new ODataV4Adaptor,
crossDomain: true
});
public eventSettings: EventSettingsModel = {
  includeFiltersInQuery: true, dataSource: this.dataManager, fields: {
    id: 'Id',
    subject: { name: 'ShipName' },
    location: { name: 'ShipCountry' },
    description: { name: 'ShipAddress' },
    startTime: { name: 'OrderDate' },
    endTime: { name: 'RequiredDate' },
    recurrenceRule: { name: 'ShipRegion' }
  }
};
}

```


MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The following image represents how the parameters are passed using ODataV4 filter.

×	Headers	Payload	Preview	Response	Initiator	Timing
▼ General						
Request URL: https://services.odata.org/V4/Northwind/Northwind.svc/Orders/?\$filter=((((OrderDate%20ge%201996-06-29T18:30:00.000Z)%20and%20(RequiredDate%20ge%201996-06-29T18:30:00.000Z))%20and%20(OrderDate%20lt%201996-08-03T18:30:00.000Z))%20or%20((OrderDate%20le%201996-06-29T18:30:00.000Z)%20and%20(RequiredDate%20gt%201996-06-29T18:30:00.000Z)))%20or%20((ShipRegion%20ne%20null)%20and%20(ShipRegion%20ne%20%27%27))						
Request Method: GET						
Status Code:  200 OK						
Remote Address: 137.117.17.70:443						
Referrer Policy: strict-origin-when-cross-origin						

Using 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 **EventID** for

the appointments by overriding the built-in response processing using the `processResponse` method of the `ODataV4Adaptor`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
class CustomAdaptor extends ODataV4Adaptor {
    override processResponse(): Object {
        let i: number = 0;
        // calling base class processResponse function
        let original: any = super.processResponse.apply(this, arguments as
any);
        // adding employee id
        original.forEach((item: any) => item['EventID'] = ++i);
        return original;
    }
}
@Component({
imports: [

        ScheduleModule,
        ButtonModule
    ],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
    // specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
    public readonly: boolean = true;
    public selectedDate: Date = new Date(1996, 6, 9);
    private dataManager: DataManager = new DataManager({
        url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Orders/',
        adaptor: new CustomAdaptor
    });
    public eventSettings: EventSettingsModel = {
        dataSource: this.dataManager, fields: {
            id: 'Id',
            subject: { name: 'ShipName' },
            location: { name: 'ShipCountry' },
            description: { name: 'ShipAddress' },
            startTime: { name: 'OrderDate' },
            endTime: { name: 'RequiredDate' },
            recurrenceRule: { name: 'ShipRegion' }
        }
    }
}
```

```
};
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

[Loading data via AJAX post](#)

You can bind the event data through external ajax request and assign it to the `dataSource` property of Scheduler. In the following code example, we have retrieved the data from server with the help of ajax request and assigned the resultant data to the `dataSource` property of Scheduler within the `onSuccess` event of Ajax.

[src/app/app.ts]

`typescript

```
import { Component, OnInit, ViewChild } from "@angular/core";
import { Ajax } from "@syncfusion/ej2-base";
import { DataManager, UrlAdaptor, Query } from "@syncfusion/ej2-data";
import { EventSettingsModel, View, EventRenderedArgs, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, ResizeService, DragAndDropService, ScheduleComponent } from
"@syncfusion/ej2-angular-schedule";
@Component({
  selector: "app-root",
  templateUrl: "app/app.component.html",
  providers: [ DayService, WeekService, WorkWeekService, MonthService, AgendaService, ResizeService,
DragAndDropService ]
})
export class AppComponent implements OnInit {
  @ViewChild("scheduleObj") public scheduleObj: ScheduleComponent;
  ngOnInit(): void {}
  public currentDate: Date = new Date(2017, 5, 5);
  onCreate() {
    const scheduleObj = this.scheduleObj; // Schedule instance
    const ajax = new Ajax(
      "https://ej2services.syncfusion.com/production/web-services/api/Schedule",
      "GET"
```

```

);
ajax.send();
ajax.onSuccess = (data: string) => {
  scheduleObj.eventSettings.dataSource = JSON.parse(data);
};
}
}
`

```

Definition for the controller method `GetData` can be referred [here](#).

Passing additional parameters to the server

To send an additional custom parameter to the server-side post, you need to make use of the `addParams` method of `Query`. Now, assign this `Query` object with additional parameters to the `query` property of Scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public readonly: boolean = true;
  public selectedDate: Date = new Date(2017, 5, 11);
  private dataManager: DataManager = new DataManager({
    url:
'https://js.syncfusion.com/demos/ejservices/api/Schedule/LoadData',
    adaptor: new ODataV4Adaptor
  });
  private dataQuery: Query = new
Query().from("Events").addParams('readOnly', 'true');

```

```
public eventSettings: EventSettingsModel = { dataSource:
this.dataManager, query: this.dataQuery };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The parameters added using the [query](#) property will be sent along with the data request sent to the server on every scheduler actions.

Handling failure actions

During the time of Scheduler interacting with server, there are chances that some server-side exceptions may occur. You can acquire those error messages or exception details in client-side using the [actionFailure](#) event of Scheduler.

The argument passed to the [actionFailure](#) event contains the error details returned from the server.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, ScheduleComponent, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule';
import { DataManager } from '@syncfusion/ej2-data';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
  AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<span style="color: #FF0000">{{err}}</span><ejs-schedule
#scheduleObj width='100%' height='530px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(actionFailure)="onActionFailure($event)"></ejs-schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2017, 5, 11);
  public err: string = '';
  private dataManager: DataManager = new DataManager({
    url: 'http://some.com/invalidUrl'
```

```

    });
    public eventSettings: EventSettingsModel = { dataSource: this.dataManager
    };
    readonly: any;
    onActionFailure(eventData: any): void {
        this.err = 'Server exception: 404 Not found';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The [actionFailure](#) event will be triggered not only on server returning errors, but also when there is an exception while processing any of the Scheduler CRUD actions.

Scheduler CRUD actions

The CRUD (Create, Read, Update and Delete) actions can be performed easily on Scheduler appointments using the various adaptors available within the **DataManager**. Most preferably, we will be using **UrlAdaptor** for performing CRUD actions on scheduler appointments.

`typescript

```

import { Component } from '@angular/core';

import { EventSettingsModel, DayService, WeekService, WorkWeekService, MonthService,
AgendaService } from '@syncfusion/ej2-angular-schedule';

import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';

@Component({
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService, AgendaService],
  // specifies the template string for the Schedule component
  template: <ejs-schedule width='100%' height='550px' [readonly]="readonly"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-schedule>
})
export class AppComponent {
  public selectedDate: Date = new Date(2017, 5, 11);
  private dataManager: DataManager = new DataManager({
    url: 'Home/GetData', // 'controller/actions'
    crudUrl: 'Home/UpdateData',
    adaptor: new UrlAdaptor
  });
}

```

```
public eventSettings: EventSettingsModel = { dataSource: this.dataManager };
}
,
```

The server-side controller code to handle the CRUD operations are as follows.

```
`c#
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using ScheduleSample.Models;
namespace ScheduleSample.Controllers
{
    public class HomeController : Controller
    {
        ScheduleDataDataContext db = new ScheduleDataDataContext();
        public ActionResult Index()
        {
            return View();
        }

        public JsonResult LoadData() // Here we get the Start and End Date and based on that can filter the data
        and return to Scheduler
        {
            var data = db.ScheduleEventDatas.ToList();
            return Json(data, JsonRequestBehavior.AllowGet);
        }

        [HttpPost]
        public JsonResult UpdateData(EditParams param)
        {
            if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
            will execute while inserting the appointments
            {
                var value = (param.action == "insert") ? param.value : param.added[0];
```

```
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
    Id = intMax + 1,
    StartTime = startTime.ToLocalTime(),
    EndTime = endTime.ToLocalTime(),
    Subject = value.Subject,
    IsAllDay = value.IsAllDay,
    StartTimezone = value.StartTimezone,
    EndTimezone = value.EndTimezone,
    RecurrenceRule = value.RecurrenceRule,
    RecurrenceID = value.RecurrenceID,
    RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
    var value = (param.action == "update") ? param.value : param.changed[0];
    var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
    if (filterData.Count() > 0)
    {
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
        appointment.StartTime = startTime.ToLocalTime();
        appointment.EndTime = endTime.ToLocalTime();
        appointment.StartTimezone = value.StartTimezone;
```



```
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}
var data = db.ScheduleEventDatas.ToList();
return Json(data, JsonRequestBehavior.AllowGet);
}
public class EditParams
{
```

```

public string key { get; set; }
public string action { get; set; }
public List<ScheduleEventData> added { get; set; }
public List<ScheduleEventData> changed { get; set; }
public List<ScheduleEventData> deleted { get; set; }
public ScheduleEventData value { get; set; }
}
}
}
,

```

Configuring Scheduler with Google API service

We have assigned our custom created Google Calendar url to the DataManager and assigned the same to the Scheduler `dataSource`. Since the events data retrieved from the Google Calendar will be in its own object format, therefore it needs to be resolved manually within the Scheduler's `dataBinding` event. Within this event, the event fields needs to be mapped properly and then assigned to the result.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { EventSettingsModel, View, DayService, WeekService, WorkWeekService,
MonthService, AgendaService, MonthAgendaService }
    from '@syncfusion/ej2-angular-schedule';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule

  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(dataBinding)="onDataBinding($event)" [readonly]="readonly"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 10, 14);
  public readonly: boolean = true;
  private calendarId: string =
'5105trob9dasha31vuqek6qgp0@group.calendar.google.com';
  private publicKey: string = 'AIzaSyD76zjMDsL_jkenM5AAnNsORypS1Icuqxg';
  private dataManger: DataManager = new DataManager({

```

```

        url: 'https://www.googleapis.com/calendar/v3/calendars/' +
        this.calendarId + '/events?key=' + this.publicKey,
        adaptor: new WebApiAdaptor,
        crossDomain: true
    });
    public eventSettings: EventSettingsModel = { dataSource: this.dataManger
};
    onDataBinding(e: { [key: string]: Object }): void {
        let items: { [key: string]: Object }[] = (e['result'] as { [key:
string]: Object; })['items'] as { [key: string]: Object }[];
        let scheduleData: Object[] = [];
        if (items.length > 0) {
            for (let i: number = 0; i < items.length; i++) {
                let event: { [key: string]: Object } = items[i];
                let when: string = (event['start'] as { [key: string]:
Object; })['dateTime'] as string;
                let start: string = (event['start'] as { [key: string]:
Object; })['dateTime'] as string;
                let end: string = (event['end'] as { [key: string]: Object;
})['dateTime'] as string;
                if (!when) {
                    when = (event['start'] as { [key: string]: Object;
})['date'] as string;
                    start = (event['start'] as { [key: string]: Object;
})['date'] as string;
                    end = (event['end'] as { [key: string]: Object;
})['date'] as string;
                }
                scheduleData.push({
                    Id: event['id'],
                    Subject: event['summary'],
                    StartTime: new Date(start),
                    EndTime: new Date(end),
                    IsAllDay: !(event['start'] as { [key: string]: Object;
})['dateTime']
                });
            }
        }
        e['result'] = scheduleData;
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler exampleLink to the Video](#) to know how to present and manipulate data.

Crud actions in Angular Schedule component

Events, a.k.a. Appointments, play an important role in Scheduler with which the users mostly interact. You can easily manipulate (add/edit/delete) the desired appointments as and when required either using the editor window or through the drag and resize action.

Add

Any kind of appointments such as normal, all-day, spanned or recurring events can be easily added on Scheduler using any one of the following ways.

- [Creation using editor window](#)
- [Creation Using addEvent method](#)

Creation using editor window

The default editor window opens when you double click on the Scheduler cells. It provides you with event related options such as Subject, Location, Start and End time, All-day, Timezone, Description and other recurrence options. With these available fields, you can choose to provide detailed information to the events. Once the fields are filled with proper values, enter the **Save** button to add an event.

In case, if you want to simply provide the Subject alone for appointments, just single click on the required cells which will open the quick popup expecting you to enter subject alone and save it. You can also select multiple cells and press **Enter** key to open the quick popup for selected time range and save the appointment for that time range.

In case, if you need to add some other additional fields to the editor window, then you can opt for [custom editor window](#) which allows you to include fields as per your application needs. If you need to add just one or two [additional fields to the existing default editor window](#), you can do so by defining it manually and then appending it to the editor window.

Creation using addEvent method

The appointments can be created dynamically by using **addEvent** method. Either you can add a single or a collection of appointment objects using **addEvent** method. The following code example let you know how to use the **addEvent** method to create multiple appointments simultaneously.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<button #addButton ej-button id="addButton" type="button"
content="Add" (click)="onButtonClick()"></button>
<ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]="scheduleViews"></ejs-schedule>`
})
export class AppComponent {
    @ViewChild("scheduleObj")
    public scheduleObj?: ScheduleComponent;
    @ViewChild("addButton")
    public addButton?: ButtonComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month'];
    public eventSettings: EventSettingsModel = {
        dataSource: [{
            Id: 1,
            Subject: 'Testing',
            StartTime: new Date(2018, 1, 11, 9, 0),
            EndTime: new Date(2018, 1, 11, 10, 0),
            IsAllDay: false
        }, {
            Id: 2,
            Subject: 'Vacation',
            StartTime: new Date(2018, 1, 13, 9, 0),
            EndTime: new Date(2018, 1, 13, 10, 0),
            IsAllDay: false
        }
    ]
    }
    public onButtonClick(): void {
        let data: Object[] = [{
            Id: 3,
            Subject: 'Conference',
            StartTime: new Date(2018, 1, 12, 9, 0),
            EndTime: new Date(2018, 1, 12, 10, 0),
            IsAllDay: true
        }, {
            Id: 4,
            Subject: 'Meeting',
            StartTime: new Date(2018, 1, 15, 10, 0),
            EndTime: new Date(2018, 1, 15, 11, 30),
            IsAllDay: false
        }
    ];
        this.scheduleObj?.addEvent(data);
        this.addButton?.element.setAttribute('disabled', 'true');
    }
}

```

MAIN.TS

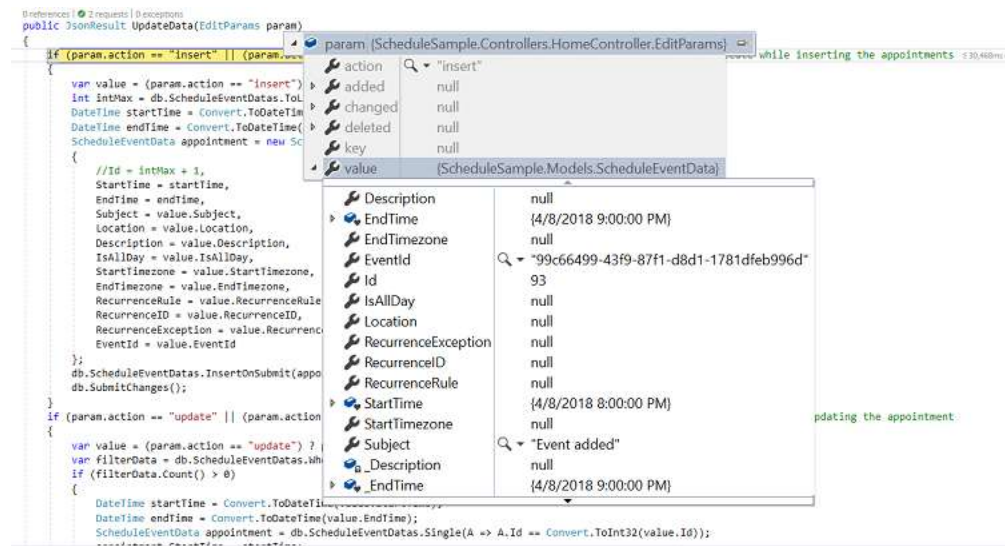
```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Inserting events into database at server-side

While adding the normal or recurring events to the Scheduler, **insert** action takes place and the following code example describes how to add a new event into database at server side.

`typescript

```
if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments
{
var value = (param.action == "insert") ? param.value : param.added[0];
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
Id = intMax + 1,
StartTime = startTime.ToLocalTime(),
EndTime = endTime.ToLocalTime(),
Subject = value.Subject,
IsAllDay = value.IsAllDay,
StartTimezone = value.StartTimezone,
EndTimezone = value.EndTimezone,
RecurrenceRule = value.RecurrenceRule,
RecurrenceID = value.RecurrenceID,
RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}
```



Restricting add action based on specific criteria

In the following example, the specific fields of Scheduler editor window such as Subject and Location are made to undergo validation such that if it is left as blank, then the default **Required** validation message will be displayed, while clicking on a save button.

Additionally, the regex condition has been added to the Location field, so that if any special characters are typed into it, then the custom validation message will be displayed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})

```

```
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
        fields: {
            subject: { name: 'Subject', validation: { required: true } },
            location: {
                name: 'Location', validation: {
                    required: true,
                    regex: ["^[a-zA-Z0-9- ]*$", 'Special character(s) not
allowed in this field']
                }
            }
        }
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

You can also dynamically prevent the creation of appointments on Scheduler. For example, say if you want to decline the creation of appointments on weekend days, you can check for its appropriate condition within the `actionBegin` event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, ActionEventArgs } from '@syncfusion/ej2-angular-
schedule';
import { scheduleData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
```



```

template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(actionBegin)="onActionBegin($event)"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public onActionBegin(args: ActionEventArgs): void {
        const weekEnds: number[] = [0, 6];
        if(args.requestType == 'eventCreate' && Array.isArray(args.data) &&
args.data.length > 0 && weekEnds.indexOf((args.data[0].StartTime).getDay())
>= 0) {
            args.cancel = true;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Edit

The same way the appointments such as normal, all-day, spanned or recurring events are created, it can be easily edited using any of the following ways.

- [Update using editor window](#)
- [Update using saveEvent method](#)

Update using editor window

You can open the default editor window filled with appointment details by double clicking on the required events. It gets pre-filled with event options such as Subject, Location, Start and End time, All-day, timezone, description and other recurrence options, from which you can edit the desired field values and, then enter the **Save** button to update it.

You can also single click on appointments, which opens the quick info popup with edit and delete options. Clicking on the **edit** option will open the default editor filled with event details and **delete** option will prompt for delete confirmation.

Updating using saveEvent method

The appointments can be edited and updated manually using the **saveEvent** method. The following code examples shows how to edit the normal and recurring events.

Normal event - Here, an event with ID **3** is edited and its subject is changed with a new text. When the modified data object is passed onto the **saveEvent** method, the changes gets reflected onto the original event. The **Id** field is mandatory in this edit process, where the modified event object should hold the valid **Id** value that exists in the Scheduler data source.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, DayService, WeekService,
WorkWeekService, MonthService, View } from '@syncfusion/ej2-angular-
schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #editButton ej2-button id="editButton" type="button"
content="Edit" (click)="onButtonClick()"></button>
  <ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]="scheduleViews"></ejs-schedule>`
})
export class AppComponent {
  @ViewChild("scheduleObj")
  public scheduleObj?: ScheduleComponent;
  @ViewChild("editButton")
  public editButton?: ButtonComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 3,
      Subject: 'Testing',
      StartTime: new Date(2018, 1, 11, 9, 0),
      EndTime: new Date(2018, 1, 11, 10, 0),
      IsAllDay: false
    }, {
      Id: 4,
      Subject: 'Vacation',
      StartTime: new Date(2018, 1, 13, 9, 0),
      EndTime: new Date(2018, 1, 13, 10, 0),
      IsAllDay: false
    }
  ]
}
  public onButtonClick(): void {
    let data: Object = {
      Id: 3,

```

```

        Subject: 'Testing-edited',
        StartTime: new Date(2018, 1, 11, 10, 0),
        EndTime: new Date(2018, 1, 11, 11, 0),
        IsAllDay: false
    };
    this.scheduleObj?.saveEvent(data);
    this.editButton?.element.setAttribute('disabled', 'true');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Recurring event - The following code example shows how to edit a single occurrence of a recurring event. In this case, the modified data should hold an additional field namely **RecurrenceID** mapping to its parent recurring event's Id value. Also, this modified occurrence will be considered as a new event in the Scheduler dataSource, where it is linked with its parent event through the **RecurrenceID** field value. The **saveEvent** method takes 2 arguments, first one accepting the modified event data object and second argument accepting either of the 2 text values - **EditOccurrence** or **EditSeries**.

When the second argument is passed as **EditOccurrence**, which means that the passed event data is a single modified occurrence - whereas if the second argument is passed as **EditSeries**, it means that the modified data needs to be edited as a whole series and therefore no new event object will be maintained in the Scheduler dataSource.

In case of modifying the single occurrence, it is also necessary to update the **RecurrenceException** field of parent event altogether with the occurrence editing. To know more about how to set **RecurrenceException** values, refer the [recurring events](#) topic.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component, ViewChild } from '@angular/core';
import { DataManager, Query, Predicate } from '@syncfusion/ej2-data';
import { ScheduleComponent, EventSettingsModel, View } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,

```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<button #editButton ejb-button id="editButton" type="button"
content="Edit" (click)="onButtonClick()"></button>
<ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]="scheduleViews"></ejs-schedule>`
})
export class AppComponent {
    @ViewChild("scheduleObj")
    public scheduleObj?: ScheduleComponent;
    @ViewChild("editButton")
    public editButton?: ButtonComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month'];
    public eventSettings: EventSettingsModel = {
        dataSource: [{
            Id: 3,
            Subject: 'Testing',
            StartTime: new Date(2018, 1, 11, 9, 0),
            EndTime: new Date(2018, 1, 11, 10, 0),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
        }, {
            Id: 4,
            Subject: 'Vacation',
            StartTime: new Date(2018, 1, 12, 11, 0),
            EndTime: new Date(2018, 1, 12, 12, 0),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
        }
    ]
    }
    public onButtonClick(): void {
        const data: Object[] = new
DataManager(this.scheduleObj?.getCurrentViewEvents()).executeLocal(new
Query().where(new Predicate('StartTime', 'lessthanorequal',
new Date(2018, 1, 11, 9, 0))));
        (data[0] as any).Subject = 'edited';
        this.scheduleObj?.saveEvent(data[0], 'EditOccurrence');
        this.editButton?.element.setAttribute('disabled', 'true');
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Updating events in database at server-side

While editing the normal events in the Scheduler, **update** action takes place and the following code example describes how to update event into database at server side.

`typescript

```
if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of code will execute while updating the appointment
```

```
{
```

```
var value = (param.action == "update") ? param.value : param.changed[0];
```

```
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
```

```
if (filterData.Count() > 0)
```

```
{
```

```
DateTime startTime = Convert.ToDateTime(value.StartTime);
```

```
DateTime endTime = Convert.ToDateTime(value.EndTime);
```

```
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id == Convert.ToInt32(value.Id));
```

```
appointment.StartTime = startTime.ToLocalTime();
```

```
appointment.EndTime = endTime.ToLocalTime();
```

```
appointment.StartTimezone = value.StartTimezone;
```

```
appointment.EndTimezone = value.EndTimezone;
```

```
appointment.Subject = value.Subject;
```

```
appointment.IsAllDay = value.IsAllDay;
```

```
appointment.RecurrenceRule = value.RecurrenceRule;
```

```
appointment.RecurrenceID = value.RecurrenceID;
```

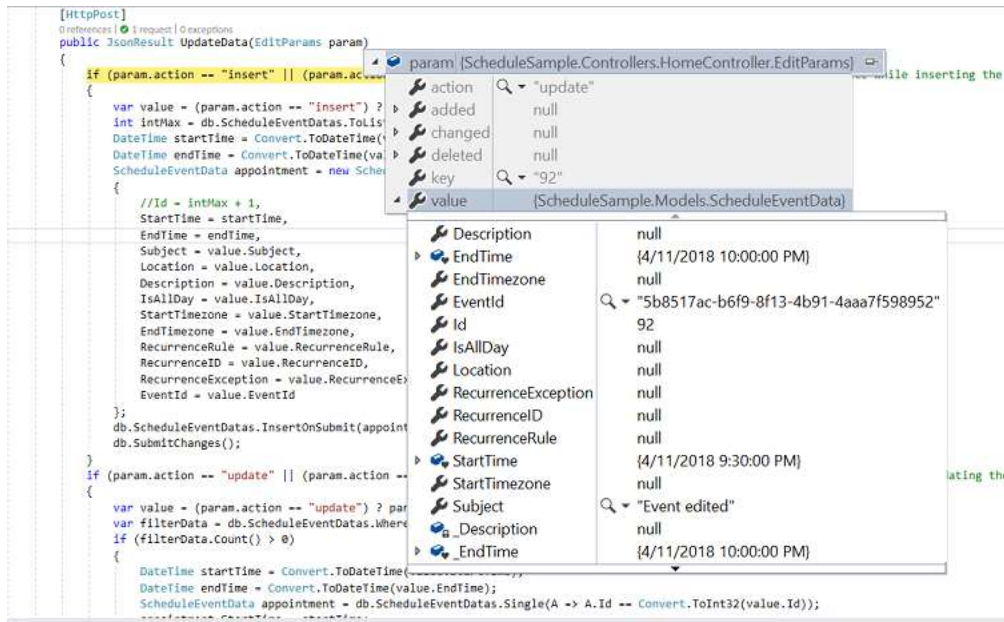
```
appointment.RecurrenceException = value.RecurrenceException;
```

```
}
```

```
db.SubmitChanges();
```

```
}
```

```
,
```



How to edit a single occurrence or entire series and update it in database at server-side

The recurring appointments can be edited in either of the following two ways.

- Single occurrence
- Entire series

Editing single occurrence - When you double click on a recurring event, a popup prompts you to choose either to edit the single event or entire series. From this, if you choose to select **EDIT EVENT** option, a single occurrence of the recurring appointment alone will be edited. The following process takes place while editing a single occurrence,

- A new event will be created from the parent event data and added to the Scheduler dataSource, with all its default field values overwritten with the newly modified data and additionally, the **recurrenceID** field will be added to it, that holds the **id** value of the parent recurring event. Also, a new **Id** will be generated for this event in the dataSource.
- The parent recurring event needs to be updated with appropriate **recurrenceException** field to hold the edited occurrence appointment's date collection.

Therefore, when a single occurrence is edited from a recurring event, the batch action takes place by allowing both the **Add** and **Edit** action requests to take place together.

In case, if you edit an existing edited occurrence of a recurring event, only those edited occurrence which present in the database as an individual event object will get updated. In this case, **update** action alone takes place on the edited occurrence object on the database.

`typescript

```

if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments

```

```

{

```

```
var value = (param.action == "insert") ? param.value : param.added[0];
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
    Id = intMax + 1,
    StartTime = startTime.ToLocalTime(),
    EndTime = endTime.ToLocalTime(),
    Subject = value.Subject,
    IsAllDay = value.IsAllDay,
    StartTimezone = value.StartTimezone,
    EndTimezone = value.EndTimezone,
    RecurrenceRule = value.RecurrenceRule,
    RecurrenceID = value.RecurrenceID,
    RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
    var value = (param.action == "update") ? param.value : param.changed[0];
    var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
    if (filterData.Count() > 0)
    {
        DateTime startTime = Convert.ToDateTime(value.StartTime);
        DateTime endTime = Convert.ToDateTime(value.EndTime);
        ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
        appointment.StartTime = startTime.ToLocalTime();
        appointment.EndTime = endTime.ToLocalTime();
    }
}
```

```

appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
`

```

Editing entire series - When you select an option **EDIT SERIES** from the popup that opens on double clicking the recurring event, the whole recurring series will be updated with the newly provided value. When this option is chosen explicitly, if a parent event holds any edited occurrences - then all its child occurrences will be removed from the dataSource and simply the single parent data will be updated.

This action of editing entire series also leads to the batch process, as both the **Delete** and **Edit** action takes place together.

`typescript

```

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
}
}

```



```

appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}

```

To know more about handling recurrence exceptions, refer the [Adding exceptions](#) topic.

How to edit from the current and following events of a series

The recurring appointments can be edited from current and following events when enable the property `editFollowingEvents`.

Editing Following Events - When you double click on a recurring event, a popup prompts you to choose either to edit the single event or Edit Following Events or entire series. From this, if you choose to select **EDIT FOLLOWING EVENTS** option, a current and following events of the recurring appointment will be edited. The following process takes place while editing a following events,

- A new event will be created from the parent event data and added to the Scheduler dataSource, with all its default field values overwritten with the newly modified data and additionally, the `followingID` field will be added to it, that holds the `id` value of the immediate parent recurring event. Also, a new `Id` will be generated for this event in the dataSource.
- The parent recurring event needs to be updated with appropriate `recurrenceRule` field to hold the modified occurrence appointment's end date.

Therefore, when a following events are edited from a recurring event, the batch action takes place by allowing the `Add`, `Edit` and `Delete` action requests to take place together.

`typescript

```
if (param.action == "insert" || (param.action == "batch" && param.added != null)) // this block of code
will execute while inserting the appointments
{
var value = (param.action == "insert") ? param.value : param.added[0];
int intMax = db.ScheduleEventDatas.ToList().Count > 0 ? db.ScheduleEventDatas.ToList().Max(p => p.Id) :
1;
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = new ScheduleEventData()
{
Id = intMax + 1,
StartTime = startTime.ToLocalTime(),
EndTime = endTime.ToLocalTime(),
Subject = value.Subject,
IsAllDay = value.IsAllDay,
StartTimezone = value.StartTimezone,
EndTimezone = value.EndTimezone,
RecurrenceRule = value.RecurrenceRule,
FollowingID = value.FollowingID,
RecurrenceID = value.RecurrenceID,
RecurrenceException = value.RecurrenceException
};
db.ScheduleEventDatas.InsertOnSubmit(appointment);
db.SubmitChanges();
}
```

if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of code will execute while updating the appointment

```
{
var value = (param.action == "update") ? param.value : param.changed[0];
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
if (filterData.Count() > 0)
{
DateTime startTime = Convert.ToDateTime(value.StartTime);
DateTime endTime = Convert.ToDateTime(value.EndTime);
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
appointment.StartTime = startTime.ToLocalTime();
appointment.EndTime = endTime.ToLocalTime();
appointment.StartTimezone = value.StartTimezone;
appointment.EndTimezone = value.EndTimezone;
appointment.Subject = value.Subject;
appointment.IsAllDay = value.IsAllDay;
appointment.RecurrenceRule = value.RecurrenceRule;
appointment.RecurrenceID = value.RecurrenceID;
appointment.FollowingID = value.FollowingID;
appointment.RecurrenceException = value.RecurrenceException;
}
db.SubmitChanges();
}
```

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of code will execute while removing the appointment

```
{
if (param.action == "remove")
{
int key = Convert.ToInt32(param.key);
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
```

```

{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}
`

```

Restricting edit action based on specific criteria

You can also dynamically prevent the editing of appointments on Scheduler. For example, say if you want to decline the updating of appointments on non-working hours, you can check for its appropriate condition within the `actionBegin` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, ActionEventArgs } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from '../datasource';
@Component({
imports: [
ScheduleModule,
ButtonModule
],
providers: [DayService,
WeekService,
WorkWeekService,
MonthService,
AgendaService,
MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(actionBegin)="onActionBegin($event)"></ejs-schedule>`
})
export class AppComponent {
@ViewChild("scheduleObj")
public scheduleObj?: ScheduleComponent;
}

```

```

public selectedDate: Date = new Date(2018, 1, 15);
public eventSettings: EventSettingsModel = { dataSource: scheduleData };
public onActionBegin(args: ActionEventArgs): void {
    if (args.requestType == 'eventChange') {
        const weekEnds: number[] = [0, 6];
        const data: { [key: string]: Object } = args.data as { [key:
string]: Object };
        const weekDay: boolean = weekEnds.indexOf((data['StartTime'] as
Date).getDay()) >= 0;
        const workHours: boolean =
((parseInt(this.scheduleObj!.workHours.start!.toString().slice(0, 2), 10) <=
(data['StartTime'] as Date).getHours()) &&
(parseInt(this.scheduleObj!.workHours.end!.toString().slice(0, 2), 10)) >=
(data['StartTime'] as Date).getHours()));
        if (weekDay || !workHours) {
            args.cancel = true;
        }
    }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Delete

The appointments can be deleted in either of the following ways,

- Selecting an appointment and clicking the delete icon from the quick popup that opens.
- Selecting an appointment and pressing Delete key.
- Selecting multiple appointments by tap holding an event and then continuously single clicking on other consecutive events and then clicking the Delete key.
- Double clicking on an event which opens the default event editor pre-filled with event details, and then choosing Delete button in it.

While performing all these above mentioned actions, a pop-up with a delete confirmation message will be displayed prompting either to proceed with deleting an appointment.

Deletion using editor window

When you double click an event, the default editor window will be opened which includes a Delete button at the bottom left position to allow you to delete that particular appointment. When deleting an appointment through this editor window, the delete alert confirmation will not be asked and the event will be deleted immediately.

Deletion using deleteEvent method

The appointments can be removed manually using the deleteEvent method. The following code examples shows how to edit the normal and recurring events.

Normal event - You can delete the normal appointments of Scheduler by simply passing its **Id** value or the entire event object collection to the **deleteEvent** method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View } from
 '@syncfusion/ej2-angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #deleteButton ej2-button id="deleteButton" type="button"
content="Delete" (click)="onButtonClick()"></button>
<ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]="scheduleViews"></ejs-schedule>`
})
export class AppComponent {
  @ViewChild("scheduleObj")
  public scheduleObj?: ScheduleComponent;
  @ViewChild("deleteButton")
  public deleteButton?: ButtonComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 3,
      Subject: 'Testing',
      StartTime: new Date(2018, 1, 11, 9, 0),
      EndTime: new Date(2018, 1, 11, 10, 0)
    }, {
      Id: 4,
      Subject: 'Vacation',
      StartTime: new Date(2018, 1, 12, 11, 0),
      EndTime: new Date(2018, 1, 12, 12, 0)
    }
  ]
}
  public onButtonClick(): void {
```

```

        this.scheduleObj?.deleteEvent(4);
        this.deleteButton?.element.setAttribute('disabled', 'true');
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Recurring Event - The recurring events can be removed as an entire series or simply removing single occurrence by using the deleteEvent method which takes in either the `DeleteSeries` or `DeleteOccurrence` parameters. The following code example shows how to delete entire series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #deleteButton ej2-button id="deleteButton" type="button"
content="Delete" (click)="onButtonClick()"></button>
<ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]="scheduleViews"></ejs-schedule>`
})
export class AppComponent {
  @ViewChild("scheduleObj")
  public scheduleObj?: ScheduleComponent;
  @ViewChild("deleteButton")
  public deleteButton?: ButtonComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
}

```

```

public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month'];
public eventSettings: EventSettingsModel = {
    dataSource: [{
        Id: 3,
        Subject: 'Testing',
        StartTime: new Date(2018, 1, 11, 9, 0),
        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3'
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 12, 11, 0),
        EndTime: new Date(2018, 1, 12, 12, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
    }]
}
public onClick(): void {
    const scheduleData: { [key: string]: Object }[] = [{
        Id: 4,
        Subject: 'Vacation',
        RecurrenceID: 4,
        StartTime: new Date(2018, 1, 12, 11, 0),
        EndTime: new Date(2018, 1, 12, 12, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
    }];
    this.scheduleObj?.deleteEvent(scheduleData, 'DeleteSeries');
    this.deleteButton?.element.setAttribute('disabled', 'true');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Removing events from database at server-side

While deleting the event from the Scheduler, **remove** action takes place and the following code example describes how to delete event from database at server side.

`typescript

```

if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
    if (param.action == "remove")
    {
        int key = Convert.ToInt32(param.key);

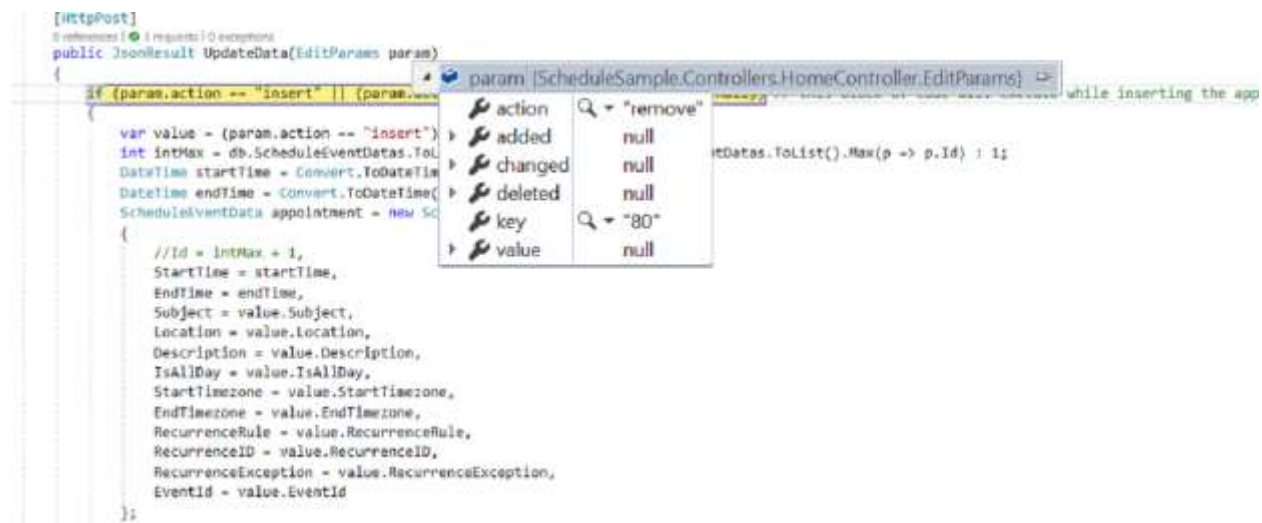
```



```

ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}

```



How to delete a single occurrence or entire series from Scheduler and update it in database at server-side
 The recurring events can be deleted in either of the following two ways.

- Single occurrence
- Entire series

Single occurrence - When you attempt to delete the recurring events, a popup prompts you to choose either to delete the single event or entire series. From this, if you choose to select **DELETE EVENT** option, a single occurrence of the recurring appointment alone will be removed. The following process takes place while removing a single occurrence,

- The selected occurrence will be deleted from the Scheduler user interface.

- In code, the parent recurring event object will be updated with appropriate `recurrenceException` field, to hold the deleted occurrence appointment's date collection.

Therefore, when a single occurrence is deleted from a recurring event, the `update` action takes place on the parent recurring event as shown in the following code example.

In case, if you delete an existing edited occurrence of a recurring event, only those edited occurrence which present in the database as an individual event object will get removed. In this case, `delete` action takes place instead of `update` action and the parent recurring event object remains same with no changes.

`typescript

```
if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of
code will execute while updating the appointment
```

```
{
```

```
var value = (param.action == "update") ? param.value : param.changed[0];
```

```
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
```

```
if (filterData.Count() > 0)
```

```
{
```

```
DateTime startTime = Convert.ToDateTime(value.StartTime);
```

```
DateTime endTime = Convert.ToDateTime(value.EndTime);
```

```
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id ==
Convert.ToInt32(value.Id));
```

```
appointment.StartTime = startTime.ToLocalTime();
```

```
appointment.EndTime = endTime.ToLocalTime();
```

```
appointment.StartTimezone = value.StartTimezone;
```

```
appointment.EndTimezone = value.EndTimezone;
```

```
appointment.Subject = value.Subject;
```

```
appointment.IsAllDay = value.IsAllDay;
```

```
appointment.RecurrenceRule = value.RecurrenceRule;
```

```
appointment.RecurrenceID = value.RecurrenceID;
```

```
appointment.RecurrenceException = value.RecurrenceException;
```

```
}
```

```
db.SubmitChanges();
```

```
}
```

```
`
```

Entire series - When you select an option **DELETE SERIES** from the popup, the whole recurring series will be deleted. When this option is chosen explicitly, if a parent event holds any edited occurrences - then

all its child occurrences which are maintained as separate event objects will also be removed from the dataSource. This action of deleting entire series leads to **remove** action and removes one or more event objects at the same time.

`typescript

```
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of
code will execute while removing the appointment
{
  if (param.action == "remove")
  {
    int key = Convert.ToInt32(param.key);
    ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
    if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
  }
  else
  {
    foreach (var apps in param.deleted)
    {
      ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
      if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
    }
  }
  db.SubmitChanges();
}
```

How to delete only the current and following events of a series

The recurring events can be deleted from current and following events only when enable **editFollowingEvents** property.

Delete Following Events - When you attempt to delete the recurring events, a popup prompts you to choose either to delete the single event or Following Events or entire series. From this, if you choose to select **FOLLOWING EVENT** option, a current and following events of the recurring appointment alone will be removed. The following process takes place while removing a single occurrence,

- The selected occurrence and the following events in same series will be deleted from the Scheduler user interface.
- In code, the parent recurring event object will be updated with appropriate **recurrenceRule** field, to update the end date of the recurring events.

Therefore, when following events are deleted from a recurring event, the **remove** and **update** action takes place on the immediate parent recurring event as shown in the following code example.

`typescript

```
if (param.action == "update" || (param.action == "batch" && param.changed != null)) // this block of code will execute while updating the appointment
```

```
{
```

```
var value = (param.action == "update") ? param.value : param.changed[0];
```

```
var filterData = db.ScheduleEventDatas.Where(c => c.Id == Convert.ToInt32(value.Id));
```

```
if (filterData.Count() > 0)
```

```
{
```

```
DateTime startTime = Convert.ToDateTime(value.StartTime);
```

```
DateTime endTime = Convert.ToDateTime(value.EndTime);
```

```
ScheduleEventData appointment = db.ScheduleEventDatas.Single(A => A.Id == Convert.ToInt32(value.Id));
```

```
appointment.StartTime = startTime.ToLocalTime();
```

```
appointment.EndTime = endTime.ToLocalTime();
```

```
appointment.StartTimezone = value.StartTimezone;
```

```
appointment.EndTimezone = value.EndTimezone;
```

```
appointment.Subject = value.Subject;
```

```
appointment.IsAllDay = value.IsAllDay;
```

```
appointment.RecurrenceRule = value.RecurrenceRule;
```

```
appointment.RecurrenceID = value.RecurrenceID;
```

```
appointment.FollowingID = value.FollowingID;
```

```
appointment.RecurrenceException = value.RecurrenceException;
```

```
}
```

```
db.SubmitChanges();
```

```
}
```

```
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) // this block of code will execute while removing the appointment
```

```
{
```

```
if (param.action == "remove")
```

```
{
```

```
int key = Convert.ToInt32(param.key);
```

```
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == key).FirstOrDefault();
```

```

if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
else
{
foreach (var apps in param.deleted)
{
ScheduleEventData appointment = db.ScheduleEventDatas.Where(c => c.Id == apps.Id).FirstOrDefault();
if (appointment != null) db.ScheduleEventDatas.DeleteOnSubmit(appointment);
}
}
db.SubmitChanges();
}
`

```

Drag and drop

When you drag and drop a normal event on the Scheduler, the event editing action takes place. When a recurring event is drag and dropped on a desired time range, the batch action explained in [Editing a single occurrence](#) process will take place - thus allowing both the [Add](#) and [Edit](#) action to take place together.

By default, when you drag a recurring instance, only the occurrence of the event gets edited and not a whole series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, DragAndDropService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from '../datasource';
@Component({
imports: [
ScheduleModule,
ButtonModule
],
providers: [DayService,
WeekService,
WorkWeekService,
MonthService,
AgendaService,
MonthAgendaService,
DragAndDropService],

```

```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Resize

When you resize a normal event on the Scheduler, the event editing action takes place. When a recurring event is resized to a new desired time, the batch action explained in [Editing a single occurrence](#) process will take place - thus allowing both the [Add](#) and [Edit](#) action to take place together.

By default, when you resize a recurring instance, only the occurrence of the event gets edited and not a whole series.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, ResizeService } from '@syncfusion/ej2-angular-
schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ResizeService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component

```

```

    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Virtual scrolling in Angular Schedule component

To achieve better performance in the Scheduler when loading a large number of resources and events, we have added virtual scrolling support in the timeline views to load a large set of resources and events instantly as you scroll. You can dynamically load large number of resources and events in timeline view of the Scheduler by setting `true` to the `allowVirtualScrolling` property within the timeline view-specific settings. The virtual loading of events is possible in Agenda view, by setting `allowVirtualScrolling` property to `true` within the agenda view specific settings.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService, TimelineYearService,
GroupModel } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService, TimelineYearService],
  standalone: true,
  selector: 'app-root',

```

```

// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' cssClass='virtual-scrolling'
height='550px' [group]="group" [selectedDate]="selectedDate"
[eventSettings]="eventSettings">
  <e-resources>
    <e-resource field='ResourceId' title='Resource'
[dataSource]='resourceDataSource' [allowMultiple]='allowMultiple'
    name='Resources' textField='Text' idField='Id' colorField='Color'>
  </e-resource>
</e-resources>
  <e-views>
    <e-view option="TimelineMonth"
[allowVirtualScrolling]="virtualscroll" isSelected=true></e-view>
    <e-view option="TimelineYear" [allowVirtualScrolling]="virtualscroll"
orientation="Vertical"></e-view>
  </e-views>
</ejs-schedule>`
))
export class AppComponent {
  public selectedDate: Date = new Date(2018, 4, 1);
  public group: GroupModel = { byGroupID: false, resources: ['Resources']
};

  public allowMultiple: boolean = true;
  public resourceDataSource: Object[] = this.generateResourceData(1, 300,
'Resource');
  public eventSettings: EventSettingsModel = { dataSource:
this.generateStaticEvents(new Date(2018, 4, 1), 300, 12) };
  public virtualscroll: boolean = true;
  private generateStaticEvents(start: Date, resCount: number, overlapCount:
number): Object[] {
    let data: Object[] = [];
    let id: number = 1;
    for (let i: number = 0; i < resCount; i++) {
      let randomCollection: number[] = [];
      let random: number = 0;
      for (let j: number = 0; j < overlapCount; j++) {
        random = Math.floor(Math.random() * (30));
        random = (random === 0) ? 1 : random;
        if (randomCollection.indexOf(random) !== -1 ||
randomCollection.indexOf(random + 2) !== -1 ||
randomCollection.indexOf(random - 2) !== -1) {
          random += (Math.max.apply(null, randomCollection) + 10);
        }
        for (let k: number = 1; k <= 2; k++) {
          randomCollection.push(random + k);
        }
        let startDate: Date = new Date(start.getFullYear(),
start.getMonth(), random);
        startDate = new Date(startDate.getTime() + (((random % 10) *
10) * (1000 * 60)));
        let endDate: Date = new Date(startDate.getTime() + ((1440 +
30) * (1000 * 60)));
        data.push({
          Id: id,
          Subject: 'Event #' + id,
          StartTime: startDate,
          EndTime: endDate,

```



```

        IsAllDay: (id % 10) ? false : true,
        ResourceId: i + 1
    });
    id++;
}
}
return data;
};
private generateResourceData(startId: number, endId: number, text:
string): Object[] {
    let data: { [key: string]: Object }[] = [];
    let colors: string[] = [
        '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
        '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
        '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a: number = startId; a <= endId; a++) {
        let n: number = Math.floor(Math.random() * colors.length);
        data.push({
            Id: a,
            Text: text + ' ' + a,
            Color: colors[n]
        });
    }
    return data;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



For now, the virtual loading of resources and events is available only `TimelineDay`, `TimelineWeek`, `TimelineWorkWeek`, `TimelineMonth` and `TimelineYear` (Vertical Orientation alone) views. In the future, we plan to port the same virtual loading on all other applicable Scheduler views.

Enabling lazy loading for appointments

The lazy loading feature provides a convenient way to efficiently load resource appointments into the Scheduler using an on-demand approach. With this feature, you can seamlessly load a large volume of appointment data into the Scheduler without experiencing any performance degradation.

By default, the Scheduler fetches all the relevant appointments from the server with in the current date range. However, enabling this feature will trigger query requests to the server for appointment retrieval whenever new resources are rendered due to scroll actions. These queries contain the resource IDs of currently displayed resources along with current date range, which can be passed as a comma-separated string. In the server controller, these resource IDs are parsed to filter the necessary appointments to render in the scheduler.

When you enable this feature, the Scheduler becomes capable of fetching events from remote services only for the current view port alone to optimize the data retrieval. The remaining appointment data is fetched from the server on-demand based on currently rendered view port resources as you scroll's through the scheduler content.

To enable this feature, you have to set the [enableLazyLoading](#) property to `true` within the view specific settings.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule, TimelineMonthService, RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
```

```

import { EventSettingsModel, GroupModel } from '@syncfusion/ej2-angular-schedule';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    RecurrenceEditorModule,
    ButtonModule

  ],
  standalone: true,
  selector: 'app-root',
  providers: [TimelineMonthService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' [readonly]='readonly'
[group]="group" [selectedDate]="selectedDate"
[eventSettings]="eventSettings">
    <e-resources>
      <e-resource field='ResourceId' title='Resource'
[dataSource]='resourceDataSource'
        name='Resources' textField='Text' idField='Id' colorField='Color'>
      </e-resource>
    </e-resources>
    <e-views>
      <e-view option="TimelineMonth" [enableLazyLoading]="enableLazyLoad"
isSelected=true></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2023, 3, 1);
  public readonly: boolean = true;
  public group: GroupModel = { resources: ['Resources'] };
  public resourceDataSource: Object[] = this.generateResourceData(1, 1000,
'Resource');
  private dataManager: DataManager = new DataManager({
    url:
'https://services.syncfusion.com/angular/production/api/VirtualEventData',
    adaptor: new WebApiAdaptor(),
    crossDomain: true
  });
  public eventSettings: EventSettingsModel = { dataSource: this.dataManager
};
  public enableLazyLoad: boolean = true;
  private generateResourceData(startId: number, endId: number, text:
string): Object[] {
    let data: { [key: string]: Object }[] = [];
    let colors: string[] = [
      '#ff8787', '#9775fa', '#748ffc', '#3bc9db', '#69db7c',
      '#fdd835', '#748ffc', '#9775fa', '#df5286', '#7fa900',
      '#fec200', '#5978ee', '#00bdae', '#ea80fc'
    ];
    for (let a: number = startId; a <= endId; a++) {
      let n: number = Math.floor(Math.random() * colors.length);
      data.push({
        Id: a,
        Text: text + ' ' + a,

```

```

        Color: colors[n]
    });
}
    return data;
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Here's the server-side controller code that retrieves appointment data based on the resource IDs provided as query parameters:

```

`c#
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.OData.Query;
namespace LazyLoadingServices.Controllers
{
    public class VirtualEventDataController : Controller
    {
        private readonly EventsContext dbContext;

        [HttpGet]
        [EnableQuery]
        [Route("api/VirtualEventData")]
        public IActionResult GetData([FromQuery] Params param)
        {
            IQueryable<EventData> query = dbContext.Events;
            // Filter the appointment data based on the ResourceId query params.
            if (!string.IsNullOrEmpty(param.ResourceId))
            {
                string[] resourceId = param.ResourceId.Split(',');
            }
        }
    }
}

```

```

query = query.Where(data => resourceId.Contains(data.ResourceId.ToString()));
}
return Ok(query.ToList());
}
}
public class Params
{
    public DateTime? StartDate { get; set; }
    public DateTime? EndDate { get; set; }
    public string ResourceId { get; set; }
}
}
`

```

Note:

- The property will be effective, when large number of resources and appointments bound to the Scheduler.
- This property is applicable only when [resource grouping](#) is enabled in Scheduler.

See Also

- [Virtual scrolling in Agenda view](#)


You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

[Editor template in Angular Schedule component](#)

Scheduler makes use of popups and dialog to display the required notifications, as well as includes an editor window with event fields for making the appointment creation and editing process easier. You can also easily customize the editor window and the fields present in it, and can also apply validations on those fields.

[Event editor](#)

The editor window usually opens on the Scheduler, when a cell or event is double clicked. When a cell is double clicked, the detailed editor window opens in "Add new" mode, whereas when an event is double clicked, the same is opened in an "Edit" mode.

In mobile devices, you can open the detailed editor window in edit mode by clicking the edit icon on the popup, that opens on single tapping an event. You can also open it in add mode by single tapping a cell, which will display a  indication, clicking on it again will open the editor window.

You can also prevent the editor window from opening, by rendering Scheduler in a **readonly** mode or by doing code customization within the **popupOpen** event.

How to change the editor window header title and text of footer buttons

You can change the header title and the text of buttons displayed at the footer of the editor window by changing the appropriate localized word collection used in the Scheduler.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { L10n } from '@syncfusion/ej2-base';
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
L10n.load({
  'en-US': {
    'schedule': {
      'saveButton': 'Add',
      'cancelButton': 'Close',
      'deleteButton': 'Remove',
      'newEvent': 'Add Event',
    },
  },
});
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, TimelineViewsService, MonthService,
AgendaService, WorkWeekService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['TimelineDay', 'Day', 'Week', 'Month',
'Agenda'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Add Event



Title

Location

|

Start

1/29/20 11:00 AM



End

1/29/20 11:30 AM



☐ All day ☐ Timezone

Repeat

Never



Description

ADD

CLOSE

[How to change the label text of default editor fields](#)

To change the default labels such as Subject, Location and other field names in the editor window, make use of the `title` property available within the field option of `eventSettings`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { WorkWeekService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, TimelineViewsService,
MonthService, AgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
```

```

providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService,
            TimelineViewsService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'TimelineWeek', 'Month',
'Agenda'];
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
        fields: {
            id: 'Id',
            subject: { name: 'Subject', title: 'Event Name' },
            location: { name: 'Location', title: 'Event Location' },
            description: { name: 'Description', title: 'Event Description' },
            startTime: { name: 'StartTime', title: 'Start Duration' },
            endTime: { name: 'EndTime', title: 'End Duration' }
        }
    };
}

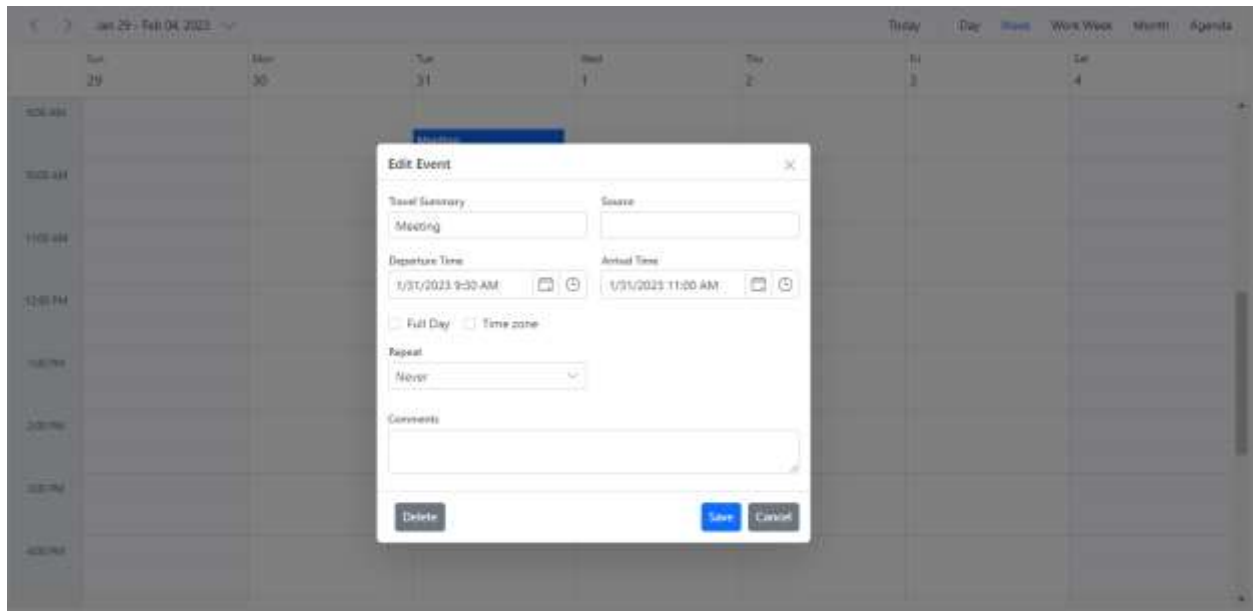
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Field validation

It is possible to validate the required fields of the editor window from client-side before submitting it, by adding appropriate validation rules to each field. The appointment fields have been extended to accept both `string` and `object` type values. To perform validations, it is necessary to specify object values for the event fields.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'></ejs-schedule>`
})
export class AppComponent {
  minValidation: (args: { [key: string]: string }) => boolean = (args: {
[key: string]: string }) => {
    return args['value'].length >= 5;
  }
}
```

```

    };
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month',
'Agenda'];
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData,
        fields: {
            id: 'Id',
            subject: { name: 'Subject', validation: { required: true } },
            location: { name: 'Location', validation: { required: true } },
            description: {
                name: 'Description', validation: {
                    required: true, minLength: [this.minValidation, 'Need
atleast 5 letters to be entered']
                }
            },
            startTime: { name: 'StartTime', validation: { required: true } },
            endTime: { name: 'EndTime', validation: { required: true } }
        }
    };
}

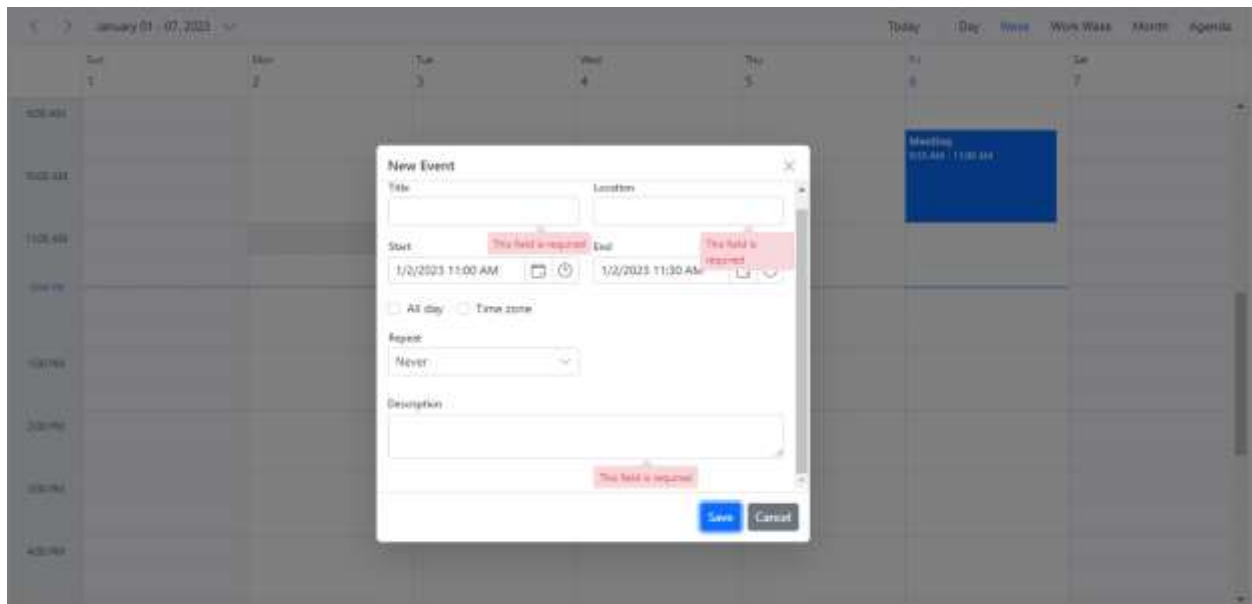
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Applicable validation rules can be referred from [form validation](#) documentation.

Add additional fields to the default editor

The additional fields can be added to the default event editor by making use of the `popupOpen` event which gets triggered before the event editor opens on the Scheduler. The `popupOpen` is a client-side

event that triggers before any of the generic popups opens on the Scheduler. The additional field (any of the form elements) should be added with a common class name `e-field`, so as to handle and process those additional data along with the default event object. In the following example, an additional field `Event Type` has been added to the default event editor and its value is processed accordingly.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { createElement } from '@syncfusion/ej2-base';
import { EventSettingsModel, PopupOpenEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { eventsData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
(popupOpen)='onPopupOpen($event)'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: eventsData
  };
  onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'Editor') {
      // Create required custom elements in initial time
      if (!args.element.querySelector('.custom-field-row')) {
        let row: HTMLElement = createElement('div', { className:
'custom-field-row' });
        let formElement: HTMLElement =
args.element.querySelector('.e-schedule-form') as HTMLElement;
        formElement.firstChild?.insertBefore(row,
args.element.querySelector('.e-title-location-row'));
        let container: HTMLElement = createElement('div', {
className: 'custom-field-container' });

```

```

        let inputEle: HTMLInputElement = createElement('input', {
            className: 'e-field', attrs: { name: 'EventType' }
        }) as HTMLInputElement;
        container.appendChild(inputEle);
        row.appendChild(container);
        let dropDownList: DropDownList = new DropDownList({
            dataSource: [
                { text: 'Public Event', value: 'public-event' },
                { text: 'Maintenance', value: 'maintenance' },
                { text: 'Commercial Event', value: 'commercial-event' },
            ],
            { text: 'Family Event', value: 'family-event' }
        ],
            fields: { text: 'text', value: 'value' },
            value: (<{ [key: string]: Object;
        })>(args.data)) ['EventType'] as string,
            floatLabelType: 'Always', placeholder: 'Event Type'
        });
        dropDownList.appendTo(inputEle);
        inputEle.setAttribute('name', 'EventType');
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to prevent the default focus of the editor widow

When we open the editor window, by default it will be focus to the Subject field. And we can able to prevent the default focusing of the editor window using the `popupOpen` event as shown in the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, PopupOpenEventArgs, ScheduleComponent } from '@syncfusion/ej2-angular-schedule';
import { eventsData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],

```

```

providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
(popupOpen)='onPopupOpen($event)'></ejs-schedule>`
})
export class AppComponent {
    @ViewChild("scheduleObj")
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public eventSettings: EventSettingsModel = {
        dataSource: eventsData
    };
    onPopupOpen(args: PopupOpenEventArgs): void {
        if (args.type === 'Editor') {
            let dialog = (args.element as any).ej2_instances[0];
            dialog.open = function(args : any) {
                this.scheduleObj.eventBase.focusElement();
            };
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing the default time duration in editor window

In default event editor window, start and end time duration are processed based on the `interval` value set within the `timeScale` property. By default, `interval` value is set to 30, and therefore the start/end time duration within the event editor will be in a 30 minutes time difference. You can change this duration value by changing the `duration` option within the `popupOpen` event as shown in the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';

```

```

import { EventSettingsModel, PopupOpenEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
(popupOpen)='onPopupOpen($event)'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'Editor') {
      args.duration = 60;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to prevent the display of editor and quick popups

You prevent the display of editor and quick popup windows by passing the value **true** to **cancel** option within the **popupOpen** event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';

```

```

import { EventSettingsModel, PopupOpenEventArgs } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
(popupOpen)='onPopupOpen($event)'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'Editor' || args.type === 'QuickInfo') {
      args.cancel = true;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

In case, if you need to prevent only specific popups on Scheduler, then you can check the condition based on the popup type. The types of the popup that can be checked within the `popupOpen` event are as follows.

Type	Description
Editor	For Detailed editor window.
QuickInfo	For Quick popup which opens on cell click.
EditEventInfo	For Quick popup which opens on event click.

- | ViewEventInfo | For Quick popup which opens on responsive mode. |
- | EventContainer | For more event indicator popup. |
- | RecurrenceAlert | For edit recurrence event alert popup. |
- | DeleteAlert | For delete confirmation popup. |
- | ValidationAlert | For validation alert popup. |
- | RecurrenceValidationAlert | For recurrence validation alert popup. |

How to prevent the display of editor in cell double click

You can prevent the display of editor using [onCellDoubleClick](#) event by setting the value `true` to `cancel` option within the event.

[src/app/app.component.ts]

```
`typescript
import { Component } from '@angular/core';
import { scheduleData } from './data';
import { extend } from '@syncfusion/ej2-base';
import { EventSettingsModel, View, DayService, WeekService, WorkWeekService, MonthService,
AgendaService, CellClickEventArgs } from '@syncfusion/ej2-angular-schedule';
@Component({
// tslint:disable-next-line:component-selector
selector: 'app-root',
template: <ejs-schedule width='100%' height='550px' [selectedDate]='selectedDate'
[views]='views' [eventSettings]='eventSettings'
(cellDoubleClick)='onCellDoubleClick($event)'></ejs-schedule>,
/ custom code end/
providers: [
DayService,
WeekService,
WorkWeekService,
MonthService,
AgendaService
],
})
export class AppComponent {
public data: Record<string, any>[] = ([] = extend([],scheduleData,null,true) as Record<string, any>[]);
public selectedDate: Date = new Date(2021, 0, 10);
```



```

public eventSettings: EventSettingsModel = { dataSource: this.data };

public currentView: View = 'Week';

public onCellDoubleClick(args: CellClickEventArgs): void {
    args.cancel = true;

    // You can use your custom dialog
}

}

`

```

Customizing timezone collection in the editor window

By default, the timezone collections in the editor window have been loaded with built-in timezone collections. Now we can be able to customize the timezone collections using the `timezoneDataSource` property with the collection of `TimezoneFields` data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService } from '@syncfusion/ej2-
angular-schedule';
@Component({
    imports: [

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService,
                TimelineMonthService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" ></ejs-
schedule>`
})
export class AppComponent {
    public data: object[] = [{
        Id: 1,
        Subject: 'Explosion of Betelgeuse Star',
        StartTime: new Date(2020, 1, 15, 10, 0),
        EndTime: new Date(2018, 1, 15, 12, 30),
        IsAllDay: false
    }

```

```

    }, {
      Id: 2,
      Subject: 'Blue Moon Eclipse',
      StartTime: new Date(2020, 1, 16, 12, 0),
      EndTime: new Date(2018, 1, 16, 13, 0),
      IsAllDay: false
    }];
    public selectedDate: Date = new Date(2020, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: this.data;
    };
    public timezoneDataSource : { Value: string, Text: string }[] = [
      { Value: 'Pacific/Niue', Text: 'Niue' },
      { Value: 'Pacific/Pago_Pago', Text: 'Pago Pago' },
      { Value: 'Pacific/Honolulu', Text: 'Hawaii Time' },
      { Value: 'Pacific/Rarotonga', Text: 'Rarotonga' },
      { Value: 'Pacific/Tahiti', Text: 'Tahiti' },
    ]
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing event editor using template

The event editor window can be customized by making use of the `editorTemplate` option. Here, the custom window design is built with the required fields using the script template and its type should be of **text/x-template**.

Each field defined within template should contain the **e-field** class, so as to allow the processing of those field values internally. The ID of this customized script template section is assigned to the `editorTemplate` option, so that these customized fields will be replaced onto the default editor window.

Note: **e-field** class only applicable for **DropDownList**, **DateTimePicker**, **MultiSelect**, **DatePicker**, **CheckBox** and **TextBox** components. Since we have processed the field values internally for the above mentioned components.

As we are using our Syncfusion sub-components within our editor using template in the following example, the custom defined form elements needs to be configured as required Syncfusion components such as **DropDownList** and [Link to the Video](#) within the `popupOpen` event. This particular step can be skipped, if the user needs to simply use the usual form elements.

Learn the easiest way to customize the editor window of Angular Scheduler with your own design by watching this video:

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'

```

```

import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ElementRef, ViewChild } from '@angular/core';
import { extend, isNullOrUndefined } from '@syncfusion/ej2-base';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { DateTimePicker } from '@syncfusion/ej2-calendars';
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { EventSettingsModel, PopupOpenEventArgs, ScheduleComponent } from
 '@syncfusion/ej2-angular-schedule';
import { eventData } from '../datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule,
    DropDownListModule,
    DateTimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo'>
    <ng-template #editorTemplate let-data>
      <table class="custom-event-editor" width="100%" cellpadding="5">
        <tbody>
          <tr>
            <td class="e-textlabel">Summary</td>
            <td colspan="4">
              <input id="Subject" class="e-field e-input"
type="text" value="" name="Subject" style="width: 100%" />
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">Status</td>
            <td colspan="4">
              <ejs-dropdownlist id='EventType' class="e-field" data-
name="EventType" placeholder='Choose Status'
[dataSource]='statusData' value="{{data.EventType}}">
            </ejs-dropdownlist>
          </td>
          </tr>
          <tr>
            <td class="e-textlabel">From</td>
            <td colspan="4">
              <ejs-datetimepicker id="StartTime" class="e-field" data-
name="StartTime" format="M/dd/yy h:mm a"
(change)="onDateChange($event)"
[value]="startDateParser(data.startTime || data.StartTime)">

```

```

        </ejs-datetimepicker>
      </td>
    </tr>
    <tr>
      <td class="e-textlabel">To</td>
      <td colspan="4">
        <ejs-datetimepicker id="EndTime" class="e-field" data-
name="EndTime" format="M/dd/yy h:mm a"
        (change)="onDateChange($event)"
[value]="endDateParser(data.endTime || data.EndTime)">
        </ejs-datetimepicker>
      </td>
    </tr>
    <tr>
      <td class="e-textlabel">Reason</td>
      <td colspan="4">
        <textarea id="Description" class="e-field e-input"
name="Description" rows="3" cols="50"
        value="{{data.Description}}" style="width: 100%; height:
60px !important; resize: vertical"></textarea>
      </td>
    </tr>
  </tbody>
</table>
</ng-template>
</ejs-schedule>`
  })
  export class AppComponent {
    @ViewChild('schedule') public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public showQuickInfo: Boolean = false;
    public startDate!: Date;
    public endDate!: Date;
    public eventSettings: EventSettingsModel = {
      dataSource: eventData
    };
    public statusData: Object[] = ['New', 'Requested', 'Confirmed'];
    public startDateParser(data: string) {
      if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {
        return new Date(data);
      } else if (!isNullOrUndefined(this.startDate)) {
        return new Date(this.startDate);
      }
      return new Date();
    }
    public endDateParser(data: string) {
      if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
        return new Date(data);
      } else if (!isNullOrUndefined(this.endDate)) {
        return new Date(this.endDate);
      }
      return new Date();
    }
    public onDateChange(args: ChangeEventArgs): void {
      if (!isNullOrUndefined(args.event as any)) {
        if (args.element.id === "StartTime") {

```

```

        this.startDate = args.value as Date;
    } else if (args.element.id === "EndTime") {
        this.endDate = args.value as Date;
    }
}
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to customize header and footer using template

The editor window's header and footer can be enhanced with custom designs using the [editorHeaderTemplate](#) and [editorFooterTemplate](#) options. To achieve this, create a script template that includes the necessary fields. Ensure that the template type is set to **text/x-template**.

In this demo, we tailor the editor's header according to the appointment's subject field using the [editorHeaderTemplate](#). Furthermore, we make use of the [editorFooterTemplate](#) to handle the functionality of validating specific fields before proceeding with the save action or canceling it if validation requirements are not met.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    ScheduleComponent, MonthService, DayService, WeekService,
    WorkWeekService, EventSettingsModel, PopupOpenEventArgs
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
        ScheduleModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    encapsulation: ViewEncapsulation.None,
    template: `
        <ejs-schedule #schedule width='100%' height='550px'
        [eventSettings]='eventSettings' (popupOpen)="onPopupOpen($event)">
        <ng-template #editorHeaderTemplate let-data>
    `
})

```

```

        <div *ngIf="data.Subject; else createNewEvent">
            {{ data.Subject }}
        </div>
        <ng-template #createNewEvent>
            Create New Event
        </ng-template>
    </ng-template>
    <ng-template #editorFooterTemplate>
        <div id="verify">
            <input type="checkbox" id="check-box" value="unchecked">
            <label id="text">Verified</label>
        </div>
        <div id="right-button">
            <button id="Save" class="e-control e-btn e-primary" disabled data-
ripple="true">Save</button>
            <button id="Cancel" class="e-control e-btn e-primary" data-
ripple="true">Cancel</button>
        </div>
    </ng-template>
</ejs-schedule>`
    })
    export class AppComponent {
        @ViewChild('schedule') public scheduleObj?: ScheduleComponent;
        private today: Date = new Date();
        private data: Record<string, any>[] = [{
            Id: 1,
            Subject: 'Surgery - Andrew',
            StartTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 9, 0),
            EndTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 10, 0),
            IsAllDay: false
        },
        {
            Id: 2,
            Subject: 'Consulting - John',
            StartTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 10, 0),
            EndTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 11, 30),
            IsAllDay: false
        },
        {
            Id: 3,
            Subject: 'Therapy - Robert',
            StartTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 11, 30),
            EndTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 12, 30),
            IsAllDay: false
        },
        {
            Id: 4,
            Subject: 'Observation - Steven',
            StartTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 12, 30),

```

```

        EndTime: new Date(this.today.getFullYear(), this.today.getMonth(),
this.today.getDate(), 13, 30),
        IsAllDay: false
    }];
    public eventSettings: EventSettingsModel = { dataSource: this.data };
    private onSaveButtonClick(args: PopupOpenEventArgs): void {
        const data: Record<string, any> = {
            Id: args.data?.['Id'],
            Subject: (args.element.querySelector('#Subject') as
HTMLInputElement).value,
            StartTime: (args.element.querySelector('#StartTime') as
any).ej2_instances[0].value,
            EndTime: (args.element.querySelector('#EndTime') as
any).ej2_instances[0].value,
            IsAllDay: (args.element.querySelector('#IsAllDay') as
HTMLInputElement).checked
        };
        if (args.target?.classList.contains('e-appointment')) {
            this.scheduleObj?.saveEvent(data, 'Save');
        } else {
            data['Id'] = this.scheduleObj?.getEventMaxID();
            this.scheduleObj?.addEvent(data);
        }
        this.scheduleObj?.closeEditor();
    }
    public onPopupOpen(args: PopupOpenEventArgs): void {
        if (args.type === 'Editor') {
            const saveButton: HTMLElement = args.element.querySelector('#Save') as
HTMLElement;
            const cancelButton: HTMLElement = args.element.querySelector('#Cancel')
as HTMLElement;
            const checkBox: HTMLInputElement = args.element.querySelector('#check-
box') as HTMLInputElement;
            checkBox.onChange = () => {
                if (!(checkBox as HTMLInputElement).checked) {
                    saveButton.setAttribute('disabled', '');
                } else {
                    saveButton.removeAttribute('disabled');
                }
            };
            saveButton.onclick = () => {
                this.onSaveButtonClick(args);
            }
            cancelButton.onclick = () => {
                this.scheduleObj?.closeEditor();
            };
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to add resource options within editor template

The resource field can be added within editor template with multiselect control for allow multiple resources.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { MultiSelect } from '@syncfusion/ej2-dropdowns';
import { DateTimePicker } from '@syncfusion/ej2-calendars';
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { isNullOrUndefined } from '@syncfusion/ej2-base'
import { ScheduleComponent, EventSettingsModel, DayService, WeekService,
WorkWeekService, MonthService, PopupOpenEventArgs, GroupModel,
AgendaService,
    MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { eventData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule,
    DateTimePickerModule,
    MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo' [group]='group'>
    <e-resources>
      <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource"
        textField="text" idField="id" colorField="color">
      </e-resource>
    </e-resources>
    <ng-template #editorTemplate let-data>
      <table class="custom-event-editor" width="100%" cellpadding="5">
        <tbody>
          <tr>
            <td class="e-textlabel">Summary</td>
            <td colspan="4">
              <input id="Subject" class="e-field e-input"
type="text" value="" name="Subject" style="width: 100%" />
            </td>
          </tr>
        </tbody>
      </table>
    </ng-template>
  `
})
```



```

        <td class="e-textlabel">From</td>
        <td colspan="4">
            <ejs-datetimepicker id="StartTime" class="e-
field" data-name="StartTime" format="M/dd/yy h:mm a"
            (change)="onDateChange($event)"
[value]="startDateParser(data.startTime || data.StartTime)">
            </ejs-datetimepicker>
        </td>
    </tr>
    <tr>
        <td class="e-textlabel">To</td>
        <td colspan="4">
            <ejs-datetimepicker id="EndTime" class="e-field"
data-name="EndTime" format="M/dd/yy h:mm a"
            (change)="onDateChange($event)"
[value]="endDateParser(data.endTime || data.EndTime)">
            </ejs-datetimepicker>
        </td>
    </tr>
    <tr>
        <td class="e-textlabel">Owner</td>
        <td colspan="4">
            <ejs-multiselect id='OwnerId'
[dataSource]='ownerDataSource' [fields]='fields' placeholder='Choose a owner'
value="{{data.OwnerId}}"></ejs-multiselect>
        </td>
    </tr>
    <tr>
        <td class="e-textlabel">Reason</td>
        <td colspan="4">
            <textarea id="Description" class="e-field e-
input" name="Description" rows="3" cols="50" style="width: 100%; height: 60px
!important; resize: vertical"></textarea>
        </td>
    </tr>
</tbody>
</table>
</ng-template>
</ejs-schedule>`
    })
    export class AppComponent {
        @ViewChild('scheduleObj')
        public scheduleObj?: ScheduleComponent;
        public selectedDate: Date = new Date(2018, 1, 15);
        public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
        public showQuickInfo: Boolean = false;
        public startDate!: Date;
        public endDate!: Date;
        public eventSettings: EventSettingsModel = {
            dataSource: eventData
        };
        public group: GroupModel = { resources: ['Owners'] };
        public ownerDataSource: Object[] = [
            { text: "Nancy", id: 1, color: "#1aaa55" },
            { text: "Smith", id: 2, color: "#7fa900" },
            { text: "Paul", id: 3, color: "#357cd2" }
        ];
    }

```

```

public fields: Object = { text: 'text', value: 'id'}
public startDateParser(data: string) {
    if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {
        return new Date(data);
    } else if (!isNullOrUndefined(this.startDate)) {
        return new Date(this.startDate);
    }
    return new Date();
}
public endDateParser(data: string) {
    if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
        return new Date(data);
    } else if (!isNullOrUndefined(this.endDate)) {
        return new Date(this.endDate);
    }
    return new Date();
}
public onChange(args: ChangeEventArgs): void {
    if (!isNullOrUndefined(args.event as any)) {
        if (args.element.id === "StartTime") {
            this.startDate = args.value as Date;
        } else if (args.element.id === "EndTime") {
            this.endDate = args.value as Date;
        }
    }
}
}

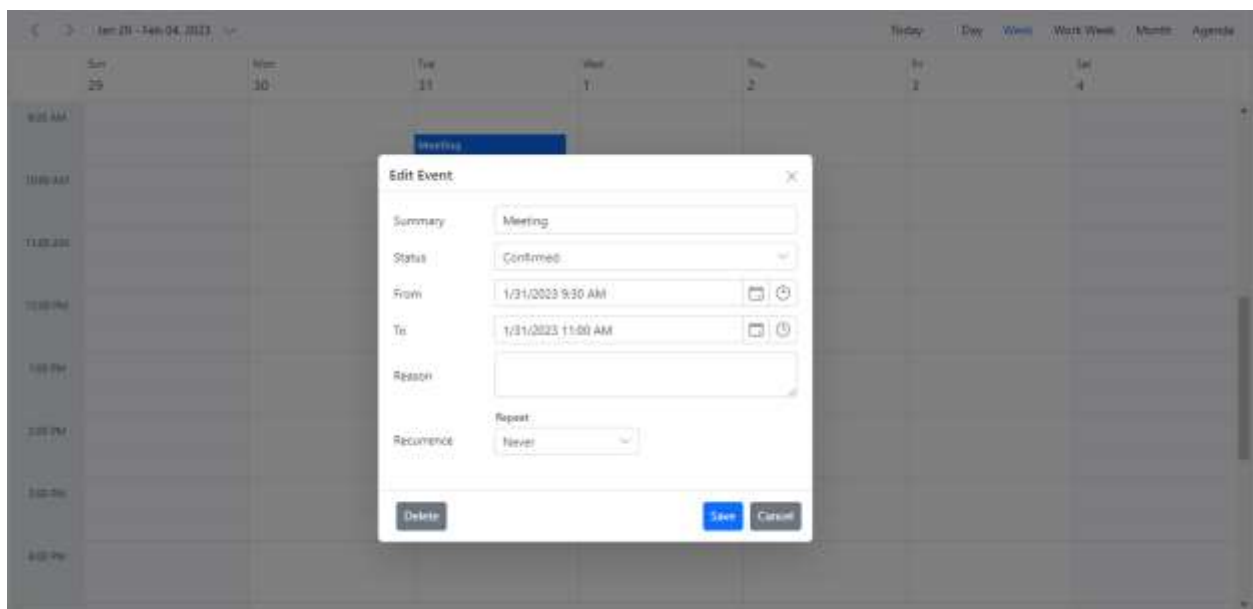
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



How to add recurrence options within editor template

The following code example shows how to add recurrence options within the editor template by importing `RecurrenceEditor`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule, RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { extend, isNullOrUndefined } from '@syncfusion/ej2-base';
import { DateTimePicker } from '@syncfusion/ej2-calendars';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, PopupOpenEventArgs, RecurrenceEditor, ScheduleComponent } from '@syncfusion/ej2-angular-schedule';
import { eventData } from '../datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule,
    RecurrenceEditorModule,
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo' (popupOpen)='onPopupOpen($event)'>
    <ng-template #editorTemplate let-data>
      <table class="custom-event-editor" width="100%" cellpadding="5">
        <tbody>
          <tr>
            <td class="e-textlabel">Summary</td>
            <td colspan="4">
              <input id="Subject" class="e-field e-input"
type="text" value="" name="Subject" style="width: 100%" />
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">From</td>
            <td colspan="4">
              <ejs-datetimepicker id="StartTime" class="e-field"
data-name="StartTime" format="M/dd/yy h:mm a"
              (change)="onDateChange($event)"
[ value]="startDateParser(data.startTime || data.StartTime)">
            </td>
          </tr>
        </tbody>
      </table>
    </ng-template>
  `
```

```

        </tr>
        <tr>
            <td class="e-textlabel">To</td>
            <td colspan="4">
                <ejs-datetimepicker id="EndTime" class="e-field"
data-name="EndTime" format="M/dd/yy h:mm a"
                (change)="onDateChange($event)"
[value]="endDateParser(data.endTime || data.EndTime)">
                </ejs-datetimepicker>
            </td>
        </tr>
        <tr>
            <td colspan="4">
                <ejs-recurrenceeditor #recurrenceObj></ejs-
recurrenceeditor>
            </td>
        </tr>
        <tr>
            <td class="e-textlabel">Reason</td>
            <td colspan="4">
                <textarea id="Description" class="e-field e-
input" name="Description" rows="3" cols="50" style="width: 100%; height: 60px
!important; resize: vertical"></textarea>
            </td>
        </tr>
    </tbody>
</table>
</ng-template>
</ejs-schedule>`
    })
    export class AppComponent {
        @ViewChild('scheduleObj')
        public scheduleObj?: ScheduleComponent;
        @ViewChild('recurrenceObj') recurrObject!: RecurrenceEditor;
        public selectedDate: Date = new Date(2018, 1, 15);
        public startDate!: Date;
        public endDate!: Date;
        public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
        public showQuickInfo: Boolean = false;
        public eventSettings: EventSettingsModel = {
            dataSource: eventData
        };
        public startDateParser(data: string) {
            if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {
                return new Date(data);
            } else if (!isNullOrUndefined(this.startDate)) {
                return new Date(this.startDate);
            }
            return new Date();
        }
        public endDateParser(data: string) {
            if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
                return new Date(data);
            } else if (!isNullOrUndefined(this.endDate)) {
                return new Date(this.endDate);
            }
            return new Date();
        }
    }

```

```

    }
    public onChangeDate(args: ChangeEventArgs): void {
        if (!isNullOrUndefined(args.event as any)) {
            if (args.element.id === "StartTime") {
                this.startDate = args.value as Date;
            } else if (args.element.id === "EndTime") {
                this.endDate = args.value as Date;
            }
        }
    }
    onPopupOpen(args: PopupOpenEventArgs): void {
        if (args.type === 'Editor') {
            if (!this.recurrObject.element.classList.contains('e-recurrenceeditor')) {
                (this.scheduleObj!.eventWindow as any).recurrenceEditor =
                this.recurrObject;
            }
            (this.recurrObject.element)!.style.display =
            (this.scheduleObj!.currentAction === "EditOccurrence") ? 'none' : 'block';
        }
    }
}

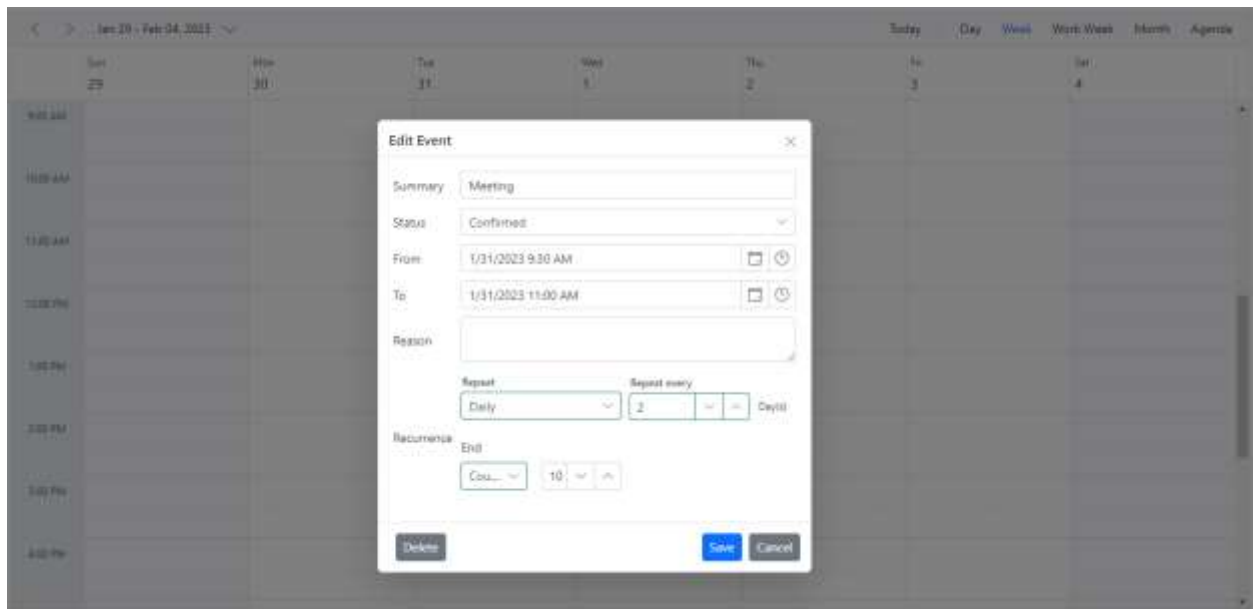
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Apply validations on editor template fields

In the following code example, validation has been added to the status field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { DateTimePicker } from '@syncfusion/ej2-calendars';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { FormValidator } from '@syncfusion/ej2-inputs';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, PopupOpenEventArgs, EJ2Instance, } from '@syncfusion/ej2-
angular-schedule';
import { eventData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule,
    DropDownListModule,
    DateTimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
  AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo' (popupOpen)='onPopupOpen($event)'>
    <ng-template #editorTemplate let-data>
      <table class="custom-event-editor" width="100%" cellpadding="5">
        <tbody>
          <tr>
            <td class="e-textlabel">Summary</td>
            <td colspan="4">
              <input id="Subject" class="e-field e-input"
type="text" value="" name="Subject" style="width: 100%" />
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">Status</td>
            <td colspan="4">
              <ejs-dropdownlist id='EventType' class="e-field"
data-name="EventType" placeholder='Choose Status'
[dataSource]='statusData' value="{{data.EventType}}"
(select)="eventSelect($event)">
            </ejs-dropdownlist>
          </td>
          </tr>
          <tr>
            <td class="e-textlabel">From</td>
            <td colspan="4">
              <ejs-datetimepicker id="StartTime" class="e-field"
data-name="StartTime" format="M/dd/yy h:mm a">

```

```

                (change)="onDateChange($event)"
[value]="startDateParser(data.startTime || data.StartTime)">
            </ejs-datetimepicker>
        </td>
    </tr>
    <tr>
        <td class="e-textlabel">To</td>
        <td colspan="4">
            <ejs-datetimepicker id="EndTime" class="e-field"
data-name="EndTime" format="M/dd/yy h:mm a"
            (change)="onDateChange($event)"
[value]="endDateParser(data.endTime || data.EndTime)">
            </ejs-datetimepicker>
        </td>
    </tr>
    <tr>
        <td class="e-textlabel">Reason</td>
        <td colspan="4">
            <textarea id="Description" class="e-field e-
input" name="Description" rows="3" cols="50"
                style="width: 100%; height: 60px !important;
resize: vertical"></textarea>
        </td>
    </tr>
</tbody>
</table>
</ng-template>
</ejs-schedule>`
    })
    export class AppComponent {
        public selectedDate: Date = new Date(2018, 1, 15);
        public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
        public showQuickInfo: Boolean = false;
        public startDate!: Date;
        public endDate!: Date;
        public statusData: Object[] = ['New', 'Requested', 'Confirmed'];
        public eventSettings: EventSettingsModel = {
            dataSource: eventData
        };
        public eventSelect(args: any) {
            if (!isNullOrUndefined(document.getElementById("EventType_Error")) as
any)) {
                document.getElementById("EventType_Error")!.style.display =
"none";
            }
        }
        public startDateParser(data: string) {
            if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {
                return new Date(data);
            } else if (!isNullOrUndefined(this.startDate)) {
                return new Date(this.startDate);
            }
            return new Date();
        }
        public endDateParser(data: string) {
            if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
                return new Date(data);
            }
        }
    }

```

```

    } else if (!isNullOrUndefined(this.endDate)) {
        return new Date(this.endDate);
    }
    return new Date();
}
public onChange(args: ChangeEventArgs): void {
    if (!isNullOrUndefined(args.event as any)) {
        if (args.element.id === "StartTime") {
            this.startDate = args.value as Date;
        } else if (args.element.id === "EndTime") {
            this.endDate = args.value as Date;
        }
    }
}
onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'Editor') {
        if
(!isNullOrUndefined(document.getElementById("EventType_Error") as any)) {
            document.getElementById("EventType_Error")!.style.display =
"none";
            document.getElementById("EventType_Error")!.style.left =
"351px";
        }
        let formElement: HTMLElement =
<HTMLElement>args.element.querySelector('.e-schedule-form');
        let validator: FormValidator = ((formElement as
EJ2Instance).ej2_instances[0] as FormValidator);
        validator.addRules('EventType', { required: true });
    }
}
}

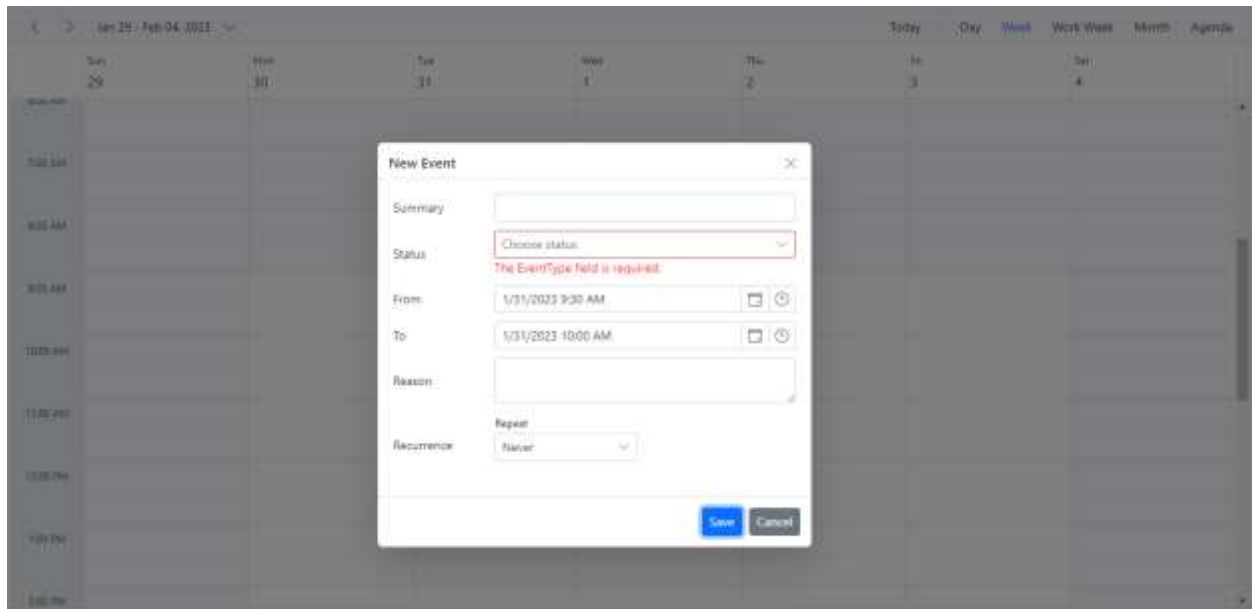
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to save the customized event editor using template

The **e-field** class is not added to each field defined within the template, so you should allow to set those field values externally by using the **popupClose** event.

Note: You can allow to retrieve the data only on the **save** and **delete** option. Data cannot be retrieved on the **close** and **cancel** options in the editor window.

The following code example shows how to save the customized event editor using a template by the **popupClose** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { RatingModule } from '@syncfusion/ej2-angular-inputs'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild, ElementRef } from '@angular/core';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { DateTimePicker } from '@syncfusion/ej2-calendars';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { RatingComponent } from '@syncfusion/ej2-angular-inputs';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, PopupOpenEventArgs, PopupCloseEventArgs, AgendaService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { eventData } from '../datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule,
    DropDownListModule,
    DateTimePickerModule,
    RatingModule
```

```

],
standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo' (popupOpen)='onPopupOpen($event)'
(popupClose)='onPopupClose($event)'>
    <ng-template #editorTemplate let-data>
      <table class="custom-event-editor" width="100%" cellpadding="5">
        <tbody>
          <tr>
            <td class="e-textlabel">Summary</td>
            <td colspan="4">
              <input id="Subject" class="e-input" type="text"
name="Subject" style="width: 100%" #subject value="{{data.Subject}}"/>
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">Rating</td>
            <td colspan="4">
              <input ejs-rating id='rating1' #rating
value="{{data.Rating}}"/>
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">From</td>
            <td colspan="4">
              <ejs-datetimepicker id="StartTime" class="e-field" data-
name="StartTime" format="M/dd/yy h:mm a"
              (change)="onDateChange($event)"
[ value]="startDateParser(data.startTime || data.StartTime)" #startTime>
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">To</td>
            <td colspan="4">
              <ejs-datetimepicker id="EndTime" class="e-field" data-
name="EndTime" format="M/dd/yy h:mm a"
              (change)="onDateChange($event)"
[ value]="endDateParser(data.endTime || data.EndTime)" #endTime>
            </td>
          </tr>
          <tr>
            <td class="e-textlabel">Reason</td>
            <td colspan="4">
              <textarea id="Description" class="e-input"
name="Description" rows="3" cols="50" style="width: 100%;
              height: 60px !important; resize: vertical" #textArea
value="{{data.Description}}"></textarea>
            </td>
          </tr>
        </tbody>
      </table>
    </ng-template>
  </div>
  `;

```

```

        </tbody>
    </table>
</ng-template>
</ejs-schedule>
})
export class AppComponent {
    @ViewChild('rating') public rating!: RatingComponent;
    @ViewChild('startTime') public startTime!: DateTimePicker;
    @ViewChild('endTime') public endTime!: DateTimePicker;
    @ViewChild('textArea') public textArea!: ElementRef<HTMLInputElement>;
    @ViewChild('subject') public subject!: ElementRef<HTMLInputElement>;
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public showQuickInfo: Boolean = false;
    public startDate!: Date;
    public endDate!: Date;
    public eventSettings: EventSettingsModel = {
        dataSource: eventData
    };
    public statusData: Object[] = ['New', 'Requested', 'Confirmed'];
    onPopupOpen(args: PopupOpenEventArgs) : void {
        if (args.type === 'Editor') {
            if (this.subject.nativeElement) {
                (this.subject.nativeElement as HTMLInputElement).value = (<{
[key: string]: Object; }>(args.data))['Subject'] as string || "";
            }
            if (this.textArea.nativeElement) {
                (this.textArea.nativeElement as HTMLInputElement).value = (<{
[key: string]: Object; }>(args.data))['Description'] as string || "";
            }
        }
    }
    onPopupClose(args: PopupCloseEventArgs) : void {
        if (args.type === 'Editor' && !isNullOrUndefined((args as any).data))
        {
            if (this.subject.nativeElement) {
                (<{ [key: string]: Object; }>(args.data))['Subject'] =
                (this.subject.nativeElement as HTMLInputElement).value;
            }
            if(this.rating.element) {
                (<{ [key: string]: Object; }>(args.data))['Rating'] as
string) = (this.rating.element as HTMLInputElement).value;
            }

            if (this.startTime.element) {
                (<{ [key: string]: Object; }>(args.data))['StartTime'] =
                (this.startTime.element as HTMLInputElement).value;
            }

            if (this.endTime.element) {
                (<{ [key: string]: Object; }>(args.data))['EndTime'] =
                (this.endTime.element as HTMLInputElement).value;
            }
            if (this.textArea.nativeElement) {
                (<{ [key: string]: Object; }>(args.data))['Description'] as
string) = (this.textArea.nativeElement as HTMLInputElement).value;
            }
        }
    }
}

```

```

    }
  }
  public startDateParser(data: string) {
    if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {
      return new Date(data);
    } else if (!isNullOrUndefined(this.startDate)) {
      return new Date(this.startDate);
    }
    return new Date();
  }
  public endDateParser(data: string) {
    if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
      return new Date(data);
    } else if (!isNullOrUndefined(this.endDate)) {
      return new Date(this.endDate);
    }
    return new Date();
  }
  public onChange(args: ChangeEventArgs): void {
    if (!isNullOrUndefined(args.event as any)) {
      if (args.element.id === "StartTime") {
        this.startDate = args.value as Date;
      } else if (args.element.id === "EndTime") {
        this.endDate = args.value as Date;
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

In case, if you need to prevent only specific popups on Scheduler, then you can check the condition based on the popup type. The types of the popup that can be checked within the `popupClose` event are as follows.

Type	Description
Editor	For Detailed editor window.
QuickInfo	For Quick popup which opens on cell click.
EditEventInfo	For Quick popup which opens on event click.
ViewEventInfo	For Quick popup which opens on responsive mode.
EventContainer	For more event indicator popup.
RecurrenceAlert	For edit recurrence event alert popup.
DeleteAlert	For delete confirmation popup.

| ValidationAlert | For validation alert popup.|

| RecurrenceValidationAlert | For recurrence validation alert popup.|

How to enable save button in customized event editor using template

Initially **e-custom-disable** class is added to the save button once all the fields are filled **e-custom-disable** class is removed from the save button.

The following code example shows how to enable save button in customized event editor using a template by the **keyup** and **change** event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewEncapsulation, ViewChild } from "@angular/core";
import { extend, isNullOrUndefined } from "@syncfusion/ej2-base";
import { ChangeEventArgs } from '@syncfusion/ej2-calendars';
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { DateTimePicker } from "@syncfusion/ej2-calendars";
import { FormValidators, FormValidator, TextBox } from "@syncfusion/ej2-angular-inputs";
import { PopupOpenEventArgs, EventRenderedArgs, ScheduleComponent,
MonthService, DayService, WeekService,
WorkWeekService, EventSettingsModel, ResizeService, DragAndDropService,
EJ2Instance , AgendaService, MonthAgendaService
} from "@syncfusion/ej2-angular-schedule";
import { eventData } from './datasource';
@Component({
imports: [

ScheduleModule,
TimePickerModule,
DropDownListModule,
DateTimePickerModule
],
standalone: true,
selector: 'app-root',
providers: [ MonthService, DayService, WeekService, WorkWeekService,
ResizeService, DragAndDropService, MonthAgendaService, AgendaService],
encapsulation: ViewEncapsulation.None,
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo' (popupOpen)='onPopupOpen($event)' >
<ng-template #editorTemplate let-data>
<table class="custom-event-editor" width="100%" cellpadding="5">
<tbody>
<tr>
<td class="e-textlabel">Summary</td>
<td colspan="4">
<input id="Subject" class=" e-field e-input"
type="text" name="Subject" style="width: 100%" value="{{data.Subject}}"
(keyup)="onChange($event)" />

```

```

        </td>
      </tr>
      <tr>
        <td class="e-textlabel">Status</td>
        <td colspan="4">
          <ejs-dropdownlist id='EventType' class="e-field" data-
name="EventType" placeholder='Choose Status '
[dataSource]='statusData' value="{{data.EventType}}">
          </ejs-dropdownlist>
        </td>
      </tr>
      <tr>
        <td class="e-textlabel">From</td>
        <td colspan="4">
          <ejs-datetimepicker id="StartTime" class="e-field" data-
name="StartTime" format="M/dd/yy h:mm a"
(change)="onDateChange($event)"
[ value]="startDateParser(data.startTime || data.StartTime)">
          </ejs-datetimepicker>
        </td>
      </tr>
      <tr>
        <td class="e-textlabel">To</td>
        <td colspan="4">
          <ejs-datetimepicker id="EndTime" class="e-field" data-
name="EndTime" format="M/dd/yy h:mm a"
(change)="onDateChange($event)"
[ value]="endDateParser(data.endTime || data.EndTime)">
          </ejs-datetimepicker>
        </td>
      </tr>
      <tr>
        <td class="e-textlabel">Reason</td>
        <td colspan="4">
          <textarea id="Description" class="e-field e-input"
name="Description" rows="3" cols="50" style="width: 100%;
height: 60px !important; resize: vertical"
value="{{data.Description}}" (keyup)="onChange($event)"></textarea>
        </td>
      </tr>
    </tbody>
  </table>
</ng-template>
</ejs-schedule>`
  })
  export class AppComponent {
    @ViewChild("scheduleObj") scheduleObj: ScheduleComponent | undefined;
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public showQuickInfo: Boolean = false;
    public startDate!: Date;
    public endDate!: Date;
    public eventSettings: EventSettingsModel = {
      dataSource: eventData,
      fields: {
        subject: { name: "Subject", validation: { required: true } },
        description: {

```

```

        name: "Description",
        validation: { required: true }
    }
}
};
public validator?: FormValidator;
public statusFields: Object = { text: "StatusText", value: "StatusText" };
public StatusData: Object[] = [
    { StatusText: "New", Id: 1 },
    { StatusText: "Requested", Id: 2 },
    { StatusText: "Confirmed", Id: 3 }
];
public onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === "Editor") {
        const formElement: HTMLElement = args.element.querySelector(".e-
schedule-form") as HTMLElement;
        this.validator = (formElement as EJ2Instance).ej2_instances[0] as
FormValidator;
        this.validator.addRules("EventType", { required: [true, "This field is
required."]});
        if (args.target!.classList.contains("e-work-cells")) {
            args.element.querySelector(".e-event-save").classList.add("e-custom-
disable");
        }
    }
}
public onChange(args : any) {
    let form = (document.querySelector(".e-schedule-form") as
any).ej2_instances[0];
    if (args.element && !args.e) {
        return;
    }
    let names = ["Subject", "Description", "EventType"];
    names.forEach(e => {
        form.validateRules(e);
    });
    let isValidated = false;
    let errorElements = document.querySelector(".e-dlg-
content").querySelectorAll(".e-schedule-error");
    for (let i = 0; i < errorElements.length; i++) {
        isValidated =(errorElements[i] as any).style.display === "none" ? true
: false;
        if (isValidated === false) {
            break;
        }
    }
    let saveBtn = document.querySelector(".e-custom-disable");
    if (isValidated && saveBtn) {
        saveBtn.classList.remove("e-custom-disable");
    } else if (!isValidated && !saveBtn) {
        document.querySelector(".e-event-save").classList.add("e-custom-
disable");
    }
}
public statusData: Object[] = ['New', 'Requested', 'Confirmed'];
public startDateParser(data: string) {
    if (isNullOrUndefined(this.startDate) && !isNullOrUndefined(data)) {

```

```

        return new Date(data);
    } else if (!isNullOrUndefined(this.startDate)) {
        return new Date(this.startDate);
    }
    return new Date();
}

public endDateParser(data: string) {
    if (isNullOrUndefined(this.endDate) && !isNullOrUndefined(data)) {
        return new Date(data);
    } else if (!isNullOrUndefined(this.endDate)) {
        return new Date(this.endDate);
    }
    return new Date();
}

public onChange(args: ChangeEventArgs): void {
    if (!isNullOrUndefined(args.event as any)) {
        if (args.element.id === "StartTime") {
            this.startDate = args.value as Date;
        } else if (args.element.id === "EndTime") {
            this.endDate = args.value as Date;
        }
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Quick popups

The quick info popups are the ones that gets opened, when a cell or appointment is single clicked on the desktop mode. On single clicking a cell, you can simply provide a subject and save it. Also, while single clicking on an event, a popup will be displayed where you can get the overview of the event information. You can also edit or delete those events through the options available in it.

By default, these popups are displayed over cells and appointments of Scheduler and to disable this action, set `false` to `showQuickInfo` property.

The quick popup that opens while single clicking on the cells are not applicable on mobile devices.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({

```



```

imports: [
    ScheduleModule,
    TimePickerModule
],
providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showQuickInfo]='showQuickInfo'></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public showQuickInfo: Boolean = false;
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to open QuickInfo popup on multiple cell selection

By default the **QuickInfo** popup will open on single click of the cell. To open the quick info popup on multiple cell selection, you need to select the cells and press **enter** key. You can open this popup immediately after multiple cell selection by setting up **true** to **quickInfoOnSelectionEnd** property where as its default value is **false**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
@Component({
imports: [
    ScheduleModule,
    TimePickerModule
],

```

```

providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' [views]='views'
[quickInfoOnSelectionEnd]='showQuickInfoOnSelectionEnd'></ejs-schedule>`
}))
export class AppComponent {
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public showQuickInfoOnSelectionEnd: Boolean = true;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to change the watermark text of quick popup subject

By default, **Add Title** text is displayed on the subject field of quick popup. To change the default watermark text, change the value of the appropriate localized word collection used in the Scheduler.

```

`typescript
L10n.load({
'en-US': {
'schedule': {
'addTitle': 'New Title'
}
}
});
`

```

Customizing quick popups

The look and feel of the built-in quick popup window, which opens when single clicked on the cells or appointments can be customized by making use of the **quickInfoTemplates** property of the Scheduler. There are 3 sub-options available to customize them easily,

- header - Accepts the template design that customizes the header part of the quick popup.
- content - Accepts the template design that customizes the content part of the quick popup.
- footer - Accepts the template design that customizes the footer part of the quick popup.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { ScheduleComponent, CurrentAction, EventSettingsModel, DayService,
WeekService, WorkWeekService, MonthService, PopupOpenEventArgs,
AgendaService,
    MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
imports: [

    ScheduleModule
],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService,
    MonthAgendaService],
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(popupOpen)="onPopupOpen($event)">
<!-- Header template -->
<ng-template #quickInfoTemplatesHeader let-data>
    <div *ngIf="data.elementType == 'cell' || data.elementType == 'event'">
        <div class="e-popup-header">
            <div class="e-header-icon-wrapper">
                <div *ngIf="data.elementType == 'event'"
class="subject">{{data.Subject}}</div>
                <button class="e-close e-close-icon e-icons" title="Close"
(click)="onCloseClick()"></button>
            </div>
        </div>
    </div>
</ng-template>
<!-- Content Template -->
<ng-template #quickInfoTemplatesContent let-data>
    <div *ngIf="data.elementType == 'cell'" class="e-cell-content">
        <form class="e-schedule-form">
            <div style="padding:10px">
                <input class="subject e-field e-input" type="text" name="Subject"
placeholder="Title" style="width:100%">
            </div>
            <div style="padding:10px">
                <input class="location e-field e-input" type="text"
name="Location" placeholder="Location" style="width:100%">
            </div>
        </form>
    </div>
    <div *ngIf="data.elementType == 'event'" class="e-event-content">
        <div class="start-time">Start:
{{data.StartTime.toLocaleString()}}</div>
        <div class="end-time">End: {{data.EndTime.toLocaleString()}}</div>
        <div *ngIf="data.Location != undefined && data.Location != ''"
class="location">Location: {{data.Location}}</div>

```

```

    </div>
  </ng-template>
  <!-- Footer Template -->
  <ng-template #quickInfoTemplatesFooter let-data>
    <div *ngIf="data.elementType == 'cell'" class="e-cell-footer">
      <div class="left-button">
        <button class="e-event-details" title="Extra Details"
(click)="onDetailsClick($event)">More Details</button>
      </div>
      <div class="right-button">
        <button class="e-event-create" title="Add"
(click)="onAddClick($event)">Add</button>
      </div>
    </div>
    <div *ngIf="data.elementType == 'event'" class="e-event-footer">
      <div class="left-button">
        <button class="e-delete" title="Delete"
(click)="onDeleteClick($event)">Delete</button>
        <button *ngIf="data.RecurrenceRule != undefined &&
data.RecurrenceRule != ''" class="e-delete-series"
title="Delete" (click)="onDeleteClick($event)">Delete
Series</button>
      </div>
      <div class="right-button">
        <button class="e-edit" title="Edit"
(click)="onEditClick($event)">Edit</button>
        <button *ngIf="data.RecurrenceRule != undefined &&
data.RecurrenceRule != ''" class="e-edit-series"
title="Edit" (click)="onEditClick($event)">Edit Series</button>
      </div>
    </div>
  </ng-template>
</ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  public selectedDate: Date = new Date(2018, 1, 15);
  private selectionTarget: Element | undefined;
  public onPopupOpen(args: PopupOpenEventArgs): void {
    this.selectionTarget = undefined;
    this.selectionTarget = args.target;
  }
  public onDetailsClick(Data: any): void {
    this.onCloseClick();
    const data: Object =
this.scheduleObj?.getCellDetails(this.scheduleObj.getSelectedElements()) as
Object;
    this.scheduleObj?.openEditor(data, 'Add');
  }
  public onAddClick(Data: any): void {
    this.onCloseClick();
  }
}

```

```

        const data: Object =
this.scheduleObj?.getCellDetails(this.scheduleObj.getSelectedElements()) as
Object;
        const eventData: { [key: string]: Object } | undefined=
this.scheduleObj?.eventWindow.getObjectFromFormData('e-quick-popup-wrapper');
        this.scheduleObj?.eventWindow.convertToEventData(data as { [key:
string]: Object }, eventData as any);
        (eventData as any)['Id'] =
this.scheduleObj?.eventBase.getEventMaxID() as number + 1;
        this.scheduleObj?.addEvent(eventData as any);
    }
    public onEditClick(args: any): void {
        if (this.selectionTarget) {
            let eventData: { [key: string]: Object } =
this.scheduleObj?.getEventDetails(this.selectionTarget) as { [key: string]:
Object };
            let currentAction: CurrentAction = 'Save';
            if (!isNullOrUndefined(eventData['RecurrenceRule']) &&
eventData['RecurrenceRule'] !== '') {
                if (args.target.classList.contains('e-edit-series')) {
                    currentAction = 'EditSeries';
                    eventData =
this.scheduleObj?.eventBase.getParentEvent(eventData, true) as any;
                } else {
                    currentAction = 'EditOccurrence';
                }
            }
            this.scheduleObj?.openEditor(eventData, currentAction);
        }
    }
    public onDeleteClick(args: any): void {
        this.onCloseClick();
        if (this.selectionTarget) {
            const eventData: { [key: string]: Object } =
this.scheduleObj?.getEventDetails(this.selectionTarget) as { [key: string]:
Object };
            let currentAction: CurrentAction = 'Delete';
            if (!isNullOrUndefined(eventData['RecurrenceRule']) &&
eventData['RecurrenceRule'] !== '') {
                currentAction = args.target.classList.contains('e-delete-series')
? 'DeleteSeries' : 'DeleteOccurrence';
            }
            this.scheduleObj?.deleteEvent(eventData, currentAction);
        }
    }
    public onCloseClick(): void {
        this.scheduleObj?.quickPopup.quickPopupHide();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The quick popup in adaptive mode can also be customized using `quickInfoTemplates` using `e-device` class.

More events indicator and popup

When the number of appointments count that lies on a particular time range * default appointment height exceeds the default height of a cell in month view and all other timeline views, a `+ more` text indicator will be displayed at the bottom of those cells. This indicator denotes that the cell contains few more appointments in it and clicking on that will display a popup displaying all the appointments present on that day.

To disable this option of showing popup with all hidden appointments, while clicking on the text indicator, you can do code customization within the `popupOpen` event.

The same indicator is displayed on all-day row in calendar views such as day, week and work week views alone, when the number of appointment count present in a cell exceeds three. Clicking on the text indicator here will not open a popup, but will allow the expand/collapse option for viewing the remaining appointments present in the all-day row.

The following code example shows how to disable the display of such popups while clicking on the more text indicator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, PopupOpenEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [currentView]='currentView'
[eventSettings]='eventSettings' (popupOpen)='onPopupOpen($event)'></ejs-
schedule>`
})
export class AppComponent {
```

```

public selectedDate: Date = new Date(2018, 1, 15);
public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
public currentView: string = 'Month';
public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
};
onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'EventContainer') {
        args.cancel = true;
    }
}
}

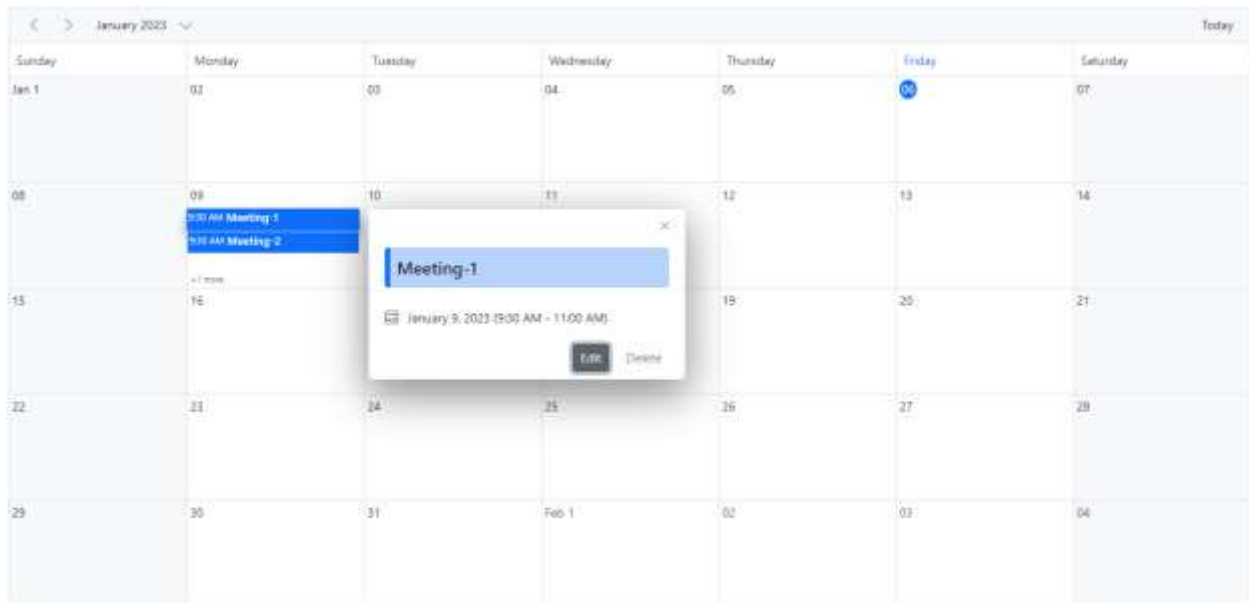
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



How to customize the popup that opens on more indicator

The following code example shows you how to customize the default more indicator popup in which number of events rendered count on the day has been shown in the header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component, ViewEncapsulation } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';

```

```

import { EventSettingsModel, PopupOpenEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [currentView]='currentView'
[cssClass]='cssClass' [eventSettings]='eventSettings'
(popupOpen)='onPopupOpen($event)'></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public currentView: string = 'Month';
  public cssClass: string = 'schedule-more-indicator';
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'EventContainer') {
      let instance: Internationalization = new Internationalization();
      let date: string = instance.formatDate((<any>args.data).date, {
skeleton: 'MMMED' });
      ((args.element.querySelector('.e-header-date')) as
HTMLElement).innerText = date;
      ((args.element.querySelector('.e-header-day')) as
HTMLElement).innerText = 'Event count: ' + (<any>args.data).event.length;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```


How to customize the appointments rendered on more indicator popup

The following code example shows you how to customize the details shown on the appointments rendered on more indicator popup.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { Internationalization, createElement } from '@syncfusion/ej2-base';
import { EventSettingsModel, PopupOpenEventArgs, ScheduleComponent } from
 '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='650px'
cssClass="more-indicator-app" [selectedDate]="selectedDate" [views]="views"
[eventSettings]="eventSettings" (popupOpen)="onPopupOpen($event)">
</ejs-schedule>`,
  styles: [`.more-indicator-app .e-more-popup-wrapper .e-appointment {
    display: inline-grid;
    height: 60px;
  }
.more-indicator-app .e-more-popup-wrapper .e-appointment .e-subject {
    white-space: initial;
  }`],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj') public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 1);
  public views: Array<string> = ['Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  private instance: Internationalization = new Internationalization();
  getTimeString(value: Date): string {
    return this.instance.formatDate(value, { format: 'hh:mm:a : dd-MMM-y'
  });
}
```

```

onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type === 'EventContainer') {
        let appointments = args.element.querySelectorAll('.e-appointment');
        for (let i = 0; i < appointments.length; i++) {
            let eventData =
this.scheduleObj?.getEventDetails(appointments[i]) as { [key: string]: Object
};
            let time = this.getTimeString(eventData['StartTime'] as Date)
+ ' - ' + this.getTimeString(eventData['EndTime'] as Date);
            let customElement = createElement('div', { className: 'e-more-popup-event-time' });
            customElement.innerText = time;
            appointments[i].insertBefore(customElement,
appointments[i].firstChild);
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to prevent the display of popup when clicking on the more text indicator

It is possible to prevent the display of popup window by passing the value **true** to **cancel** option within the **MoreEventsClick** event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, MoreEventsClickArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
})

```

```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [currentView]='currentView'
[eventSettings]='eventSettings'
(moreEventsClick)='onMoreEventsClick($event)'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public currentView: string = 'Month';
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  onMoreEventsClick(args: MoreEventsClickArgs): void {
    args.cancel = true;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to navigate Day view when clicking on more text indicator

The following code example shows you how to customize the `moreEventsClick` property to navigate to the Day view when clicking on the more text indicator.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, MoreEventsClickArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
})

```

```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [currentView]='currentView'
[eventSettings]='eventSettings'
(moreEventsClick)='onMoreEventsClick($event)'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public currentView: string = 'Month';
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  onMoreEventsClick(args: MoreEventsClickArgs): void {
    args.isPopupOpen = false;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to close the editor window manually

You can close the editor window by using [closeEditor](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',

```

```
// specifies the template string for the Schedule component
template: `<button ej-button cssClass= 'e-custom-close'
(click)='closeEditor()'> closeEditor </button> <ejs-schedule #scheduleObj
width='100%' height='650px' cssClass="more-indicator-app"
[selectedDate]="selectedDate" [views]="views"
[eventSettings]="eventSettings"></ejs-schedule>`,
})
export class AppComponent {
  @ViewChild('scheduleObj') public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2023, 2, 5);
  public views: Array<string> = ['Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 1,
      Subject: 'Review Meeting',
      StartTime: new Date(2023, 2, 5, 20, 0, 0),
      EndTime: new Date(2023, 2, 5, 21, 0, 0)
    }]
  };
  closeEditor(): void {
    this.scheduleObj?.closeEditor();
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

How to open the quick info popup manually

You can open the quick info popup in scheduler by using the [openQuickInfoPopup](#) public method. To open the cell quick info popup, you can pass the cell data as an argument to the method. To open the event quick info popup, you should pass the event data object as an argument to the method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<button ejs-button cssClass= 'e-custom-close'
(click)='cellClick()'> Show Cell Click Popup </button>
    <button ejs-button cssClass= 'e-custom-close' (click)='eventClick()'> Show
Event Click Popup </button>
    <ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]='selectedDate' [eventSettings]='eventSettings' > <e-views> <e-
view option="Week"></e-view> <e-view option="WorkWeek"></e-view> <e-view
option="Month"></e-view> <e-view option="Day"></e-view> </e-views> </ejs-
schedule>`
    })
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2023, 2, 5);
    public eventSettings: EventSettingsModel = {
        dataSource: [{
            Id: 1,
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        }
    ]
    };
    cellClick(): void {
        let cellData: Object = {
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        this.scheduleObj?.openQuickInfoPopup(cellData);
    }
    eventClick(): void {
        let eventData: Object = {
            Id: 1,
            Subject: 'Review Meeting',
            StartTime: new Date(2023, 2, 5, 9, 0, 0),
            EndTime: new Date(2023, 2, 5, 10, 0, 0)
        };
        this.scheduleObj?.openQuickInfoPopup(eventData);
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to close the quick info popup manually

You can close the quick info popup in scheduler by using the [closeQuickInfoPopup](#) public method. The following code example demonstrates the how to close quick info popup manually.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button ej2-button cssClass= 'e-custom-close'
(click)="CloseQuickInfoPopup">CloseQuickInfoPopup</button> <ejs-schedule
#scheduleObj width='100%' height='650px' cssClass="more-indicator-app"
[selectedDate]="selectedDate" [views]="views"
[eventSettings]="eventSettings"></ejs-schedule>`,
})
export class AppComponent {
  @ViewChild('scheduleObj') public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2023, 2, 5);
  public views: Array<string> = ['Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 1,
      Subject: 'Review Meeting',
      StartTime: new Date(2023, 2, 5, 20, 0, 0),
      EndTime: new Date(2023, 2, 5, 21, 0, 0)
    }]
  };
  CloseQuickInfoPopup(): void {
    this.scheduleObj?.closeQuickInfoPopup();
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Timezone in Angular Schedule component

The Scheduler 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. Apart from the default action of applying specific timezone to the Scheduler, it is also possible to set different time zone values for each appointments through the properties `startTimezone` and `endTimezone` which can be defined as separate fields within the event fields collection.

Note: `timezone` property only applicable for the appointment processing and current time indication.

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.

Scheduler with no timezone

When no specific time zone is set to Scheduler, appointments will be displayed based on the client system's timezone which is the default behavior. Here, the same appointment when viewed from different timezone will have different start and end times.

The following code example displays an appointment from 9.00 AM to 10.00 AM when you open the Scheduler from any of the timezone. This is because, we are providing the start and end time enclosing with `new Date()` which works based on the client browser's timezone.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService } from '@syncfusion/ej2-
angular-schedule';
import { fifaEventsData } from './datasource';
import { extend } from '@syncfusion/ej2-base';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
```



```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Day"></e-view> <e-view option="Week"></e-view> <e-view
option="TimelineWeek"></e-view> <e-view option="Month"></e-view> <e-view
option="Agenda"></e-view> </e-views> </ejs-schedule>`
))
export class AppComponent {
    public eventSettings: EventSettingsModel = {
        dataSource: [{
            Id: 1,
            Subject: 'Paris',
            StartTime: new Date(2018, 1, 15, 9, 0),
            EndTime: new Date(2018, 1, 15, 10, 0)
        }]
    };
    public selectedDate: Date = new Date(2018, 1, 15);
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Scheduler set to specific timezone

When a time zone is set to Scheduler through `timezone` property, the appointments will be displayed exactly based on the Scheduler timezone regardless of its client timezone. In the following code example, appointments will be displayed based on Eastern Time (UTC -05:00).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { timeZoneData } from './datasource';
import { extend } from '@syncfusion/ej2-base';
@Component({
imports: [

        ScheduleModule,
        ButtonModule

```

```

    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" timezone="America/New_York"
[eventSettings]="eventSettings" > <e-views> <e-view option="Day"></e-view>
<e-view option="Week"></e-view> <e-view option="Month"></e-view> </e-views>
</ejs-schedule>`
  })
  export class AppComponent {
    public eventSettings: EventSettingsModel = {
      dataSource: <Object[]>extend([], timeZoneData, undefined, true)
    };
    public selectedDate: Date = new Date(2018, 1, 17);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Display events on same time everywhere with no time difference

Setting timezone to UTC for Scheduler will display the appointments on same time as in the database for all the users in different time zone.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, Timezone } from '@syncfusion/ej2-angular-
schedule';
import { fifaEventsData } from './datasource';
import { extend } from '@syncfusion/ej2-base';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
              WeekService,

```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" timezone="UTC" [eventSettings]="eventSettings"
> <e-views> <e-view option="Day"></e-view> <e-view option="Week"></e-view>
<e-view option="Month"></e-view> </e-views> </ejs-schedule>`
})
export class AppComponent {
    public fifaEvents: Object[] = <Object[]>extend([], fifaEventsData,
undefined, true);
    public selectedDate: Date = new Date(2018, 5, 17);
    public eventSettings?: EventSettingsModel;
    constructor() {
        let timezone: Timezone = new Timezone();
        for (let fifaEvent of this.fifaEvents) {
            let event: { [key: string]: Object } = fifaEvent as { [key:
string]: Object };
            event['StartTime'] =
timezone.removeLocalOffset(<Date>event['StartTime']);
            event['EndTime'] =
timezone.removeLocalOffset(<Date>event['EndTime']);
        }
        this.eventSettings = { dataSource: this.fifaEvents };
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set specific timezone for events

It is possible to set different timezone for Scheduler events by setting `startTimezone` and `endTimezone` properties within the `eventSettings` option. It allows each appointment to maintain different timezone and displays on Scheduler with appropriate time differences.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { fifaEventsData } from './datasource';
import { extend } from '@syncfusion/ej2-base';

```

```

@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Day"></e-view> <e-view option="Week"></e-view> <e-view
option="Month"></e-view> </e-views> </ejs-schedule>`
})
export class AppComponent {
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 1,
      Subject: 'Paris',
      StartTime: new Date(2018, 1, 15, 10, 0),
      EndTime: new Date(2018, 1, 15, 12, 30),
      StartTimezone: 'Europe/Moscow',
      EndTimezone: 'Europe/Moscow'
    }]
  };
  public selectedDate: Date = new Date(2018, 1, 15);
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Add or remove timezone names to/from the timezone collection

Instead of displaying all the timezone names within the timezone collection (more than 200 are displayed on the editor window timezone fields by default), you can customize the timezone collection at application end as shown in the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';

```

```

import { EventSettingsModel, timezoneData } from '@syncfusion/ej2-angular-schedule';
import { timeZoneData } from './datasource';
import { extend } from '@syncfusion/ej2-base';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-view option="Day"></e-view> <e-view option="Week"></e-view> <e-view option="Month"></e-view> </e-views> </ejs-schedule>`
})
export class AppComponent {
  public eventSettings: EventSettingsModel = {
    dataSource: <Object[]>extend([], timeZoneData, undefined, true)
  };
  public selectedDate: Date = new Date(2018, 1, 11);
  private timeZones: { Text: string, Value: string }[] = [
    { Value: 'America/New_York', Text: '(UTC-05:00) Eastern Time' },
    { Value: 'UTC', Text: 'UTC' },
    { Value: 'Asia/Kolkata', Text: '(UTC+05:30) India Standard Time' }
  ];
  constructor() {
    timezoneData.splice(0, timezoneData.length, ...this.timeZones);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

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**

`typescript

// 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**

`typescript

// 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

,

add

This method is used to add the time difference between passed UTC date and timezone.

| Parameters | Type | Description |

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

| Date | Date | UTC time as date object. |

| Timezone | String | Timezone. |

Returns **Date**

`typescript

```
// 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.add(date, "Europe/Paris");
console.log(convertedDate); //2018-12-05T05:25:11.000Z
`
```

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**

`typescript

```
// 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
`
```

removeLocalOffset

This method is used to remove the local offset time from the date passed.

Parameters	Type	Description
----- ----- -----		
Date	Date	UTC as date object.

Returns **Date**

`typescript

```
// 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.removeLocalOffset(date);
console.log(convertedDate); //2018-12-05T15:25:11.000Z
`
```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Views in Angular Schedule component

The Scheduler includes wide variety of view modes with unique configuration options for each view. The available view modes are Day, Week, Work Week, Month, Year, Agenda, Month Agenda, Timeline Day, Timeline Week, Timeline Work Week and Timeline Month, out of which the [Link to the Video](#) view is set as active.

To navigate between different views and dates, the navigation options are available at the Scheduler header bar. The active view option is usually highlighted by default. The date range of the active view will also be displayed at the left corner of the header bar, clicking on which will open a calendar popup for the ease of desired date selection.

Learn how to customize each individual view of Angular Scheduler with different settings by watching this video:

By default, Scheduler displays the calendar views such as day, week, work week, month and agenda.

Setting specific view on scheduler

As the Scheduler displays **week** view by default, therefore to change the active view, set **currentView** property with the desired view name. The applicable view names that the Scheduler accepts are as follows,

- Day
- Week
- WorkWeek
- Month
- Year
- Agenda
- MonthAgenda
- TimelineDay
- TimelineWeek
- TimelineWorkWeek
- TimelineMonth
- TimelineYear

It is necessary to import and inject the appropriate view modules into the application to make use of these view modes on the Scheduler. Also, it is possible to display only the desired views on the Scheduler. To define and configure specific views, use the [views](#) property.

In the following example, the Scheduler displays 4 views namely, Week, Month, TimelineWeek and Timeline Month. The appropriate view modules are imported and injected properly to display those views on the Scheduler.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
```



```

import { DayService, WorkWeekService, AgendaService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, WeekService, TimelineViewsService, MonthService,
 TimelineMonthService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    TimelineViewsService, MonthService, TimelineMonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='TimelineDay'></e-view>
      <e-view option='Week'></e-view>
      <e-view option='Month'></e-view>
      <e-view option='TimelineMonth'></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

To configure Scheduler with simply 2 views, but with different configurations on each view, refer the following code example. Here, the Week view displays the dates in **dd-MM-yyyy** format whereas the Month view hides the weekend days and also displays it in readonly mode.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
 AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'

```

```

import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
  <e-views>
    <e-view option='Week' dateFormat='dd-MMM-yyyy'></e-view>
    <e-view option='Month' [showWeekend]="showWeekend"
[readonly]="isReadOnly"></e-view>
  </e-views>
</ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public showWeekend: Boolean = true;
  public isReadOnly: number = 3;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

View specific configuration

There are scenarios where each view may need to have different configurations. For such cases, you can define the applicable scheduler properties within the **views** Property for each view option as depicted in the following examples. The fields available to be used within each view options are as follows.

Property	Type	Description	Applicable views
option	View	It accepts the Scheduler view name, based on which we can define its related properties. The view names can be Day , Week and so on.	All views.
isSelected	Boolean	It acts similar to the currentView property and defines the active view of the Scheduler.	All views.

| **dateFormat** | Date | By default, Scheduler follows the date format as per the default culture assigned to it. When it is defined under specific view, only those assigned views follows this date format. | All views. |

| **readonly** | Boolean | When set to **true**, prevents the CRUD actions on the respective view under where it is defined. | All views. |

| **resourceHeaderTemplate** | String | The template option which is used to customize the resource header cells on the Scheduler. It gets applied only on the views, wherever it is defined. | All views. |

| **dateHeaderTemplate** | String | The template option which is used to customize the date header cells and is applied only on the views, wherever it is defined. | All views. |

| **eventTemplate** | String | The template option to customize the events background. It will get applied to the events of the view to which it is currently being defined. | All views. |

| **showWeekend** | Boolean | When set to **false**, it hides the weekend days of a week from the views on which it is defined. | All views. |

| **group** | [GroupModel](#) | Allows to set different resource grouping options on all available Scheduler view modes. | All views. |

| **cellTemplate** | String | The template option to customize the work cells of the Scheduler and is applied only on the views, on which it is defined. | Applicable on all views except Agenda view. |

| **workDays** | Number[] | It is used to set the working days on the Scheduler views. | Applicable on all views except Agenda view. |

| **displayName** | String | When a particular view is customized to display with different intervals, this property allows the user to set different display name for each of the views. | Applicable on all views except Agenda and Month Agenda. |

| **interval** | Number | It allows to customize the default Scheduler views with different set of days, weeks, work weeks or months on the applicable view type. | Applicable on all views except Agenda and Month Agenda. |

| **startHour** | String | It is used to specify the start hour, from which the Scheduler should be displayed. It accepts the time string in a short skeleton format and also, hides the time beyond the specified start time. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week and Timeline Work Week views. |

| **endHour** | String | It is used to specify the end hour, at which the Scheduler ends. It accepts the time string in a short skeleton format. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week, and Timeline Work Week views. |

| **timeScale** | [TimeScaleModel](#) | Allows to set different timescale configuration on each applicable view modes. | Applicable on Day, Week, Work Week, Timeline Day, Timeline Week, and Timeline Work Week views. |

| **showWeekNumber** | Boolean | When set to **true**, shows the week number on the respective weeks. | Applicable on Day, Week, Work Week, and Month views. |

| **allowVirtualScrolling** | Boolean | It is used to enable or disable the virtual scrolling functionality. | Applicable on Agenda and Timeline views. |

| **headerRows** | [HeaderRowsModel](#) | Allows defining the custom header rows on timeline views of the Scheduler to display the year, month, week, date and hour label as an individual row. | Applicable only on all timeline views. |

Day view

Usually a day view displays a single day with all its related appointments. It is possible to customize the day view to display more number of days by extending the **views** property with **interval** option. You can also define any of the above defined properties within the **views** object definition as depicted in the following code example.

APP.COMPONENT.TS

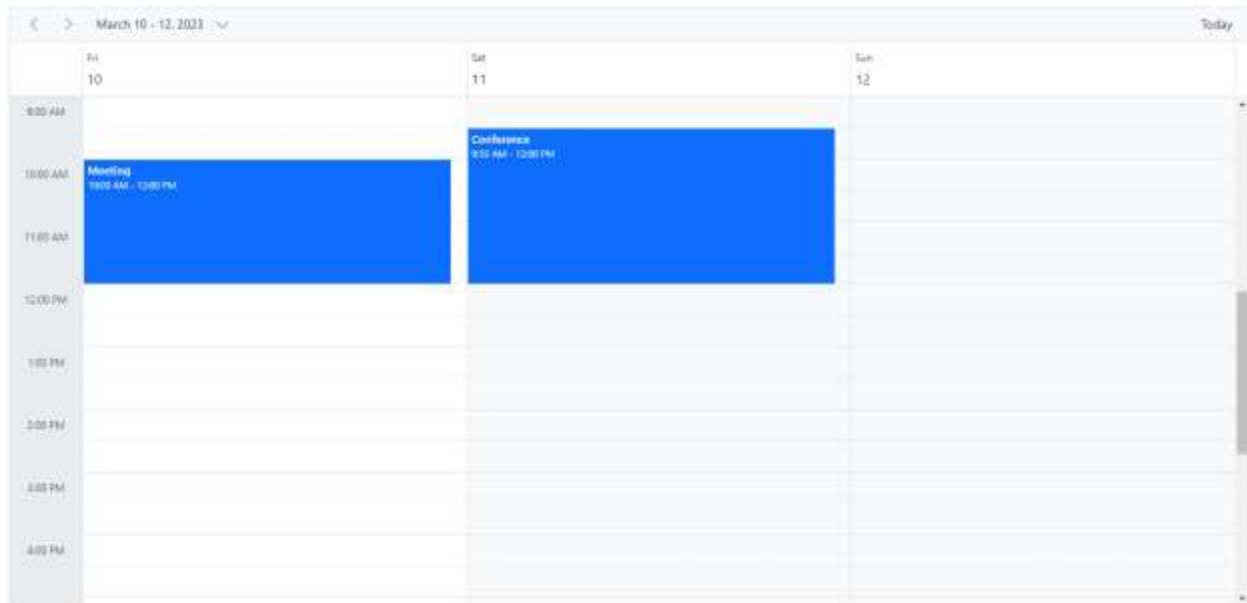
```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' currentView="Day"
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='Day' startHour='09:30' endHour='18:00'
[timeScale]="timeScaleOptions"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public timeScaleOptions: TimeScaleModel = { enable: true, slotCount: 5 };
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



All the above defined properties can be accessed within Day view except `allowVirtualScrolling` and `headerRows`.

Week view

The Week view displays a count of 7 days (from Sunday to Saturday) with all its related appointments. The first day of the week can be changed using the `firstDayOfWeek` which accepts the integer (Sunday=0, Monday=1, Tuesday=2 and so on) value. You can navigate to a particular date in day view from the week view by clicking on the appropriate dates on the date header bar.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
```

```

standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' currentView="Week"
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
  <e-views>
    <e-view option='Day' [interval]="viewInterval" displayName='2 Days'
startHour='09:30' endHour='18:00' [timeScale]="timeScaleOptions"></e-view>
    <e-view option='Week' [interval]="viewInterval" displayName='2 Weeks'
[showWeekend]="showWeekend" [isSelected]="isSelected"></e-view>
  </e-views>
</ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public viewInterval: number = 2;
  public showWeekend: boolean = false;
  public isSelected: boolean = true;
  public timeScaleOptions: TimeScaleModel = { enable: true, slotCount: 5 };
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

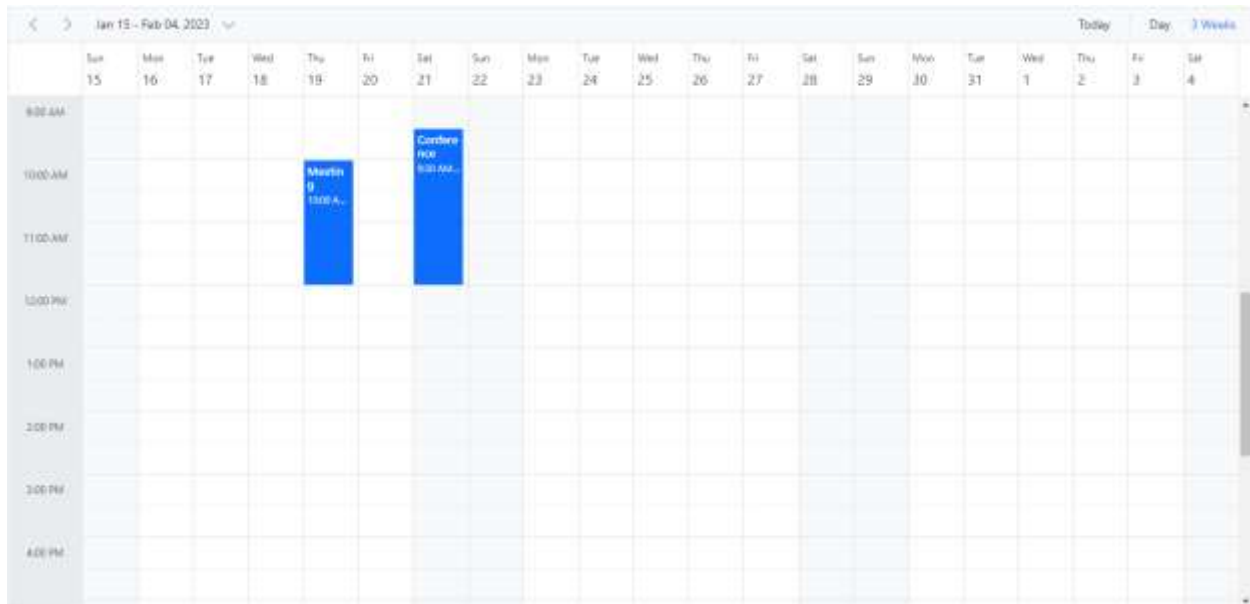
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



All the above defined properties in the table can be accessed within Week and Work week views except `allowVirtualScrolling` and `headerRows`.

Work Week view

The Work week view displays only the working days of a week (count of 5 days) and its associated appointments. It is possible to customize the working days on the work week view by using the **workDays** property which accepts an array of integer values (such as Sunday=0, Monday=1, Tuesday=2 and so on). By default, it displays from Monday to Friday (5 days). You can also navigate to a particular date in the day view from the work week view by clicking on the appropriate dates in the date header bar.

The following code example depicts how to change the working days only on the **Work Week** view of the Scheduler.

APP.COMPONENT.TS

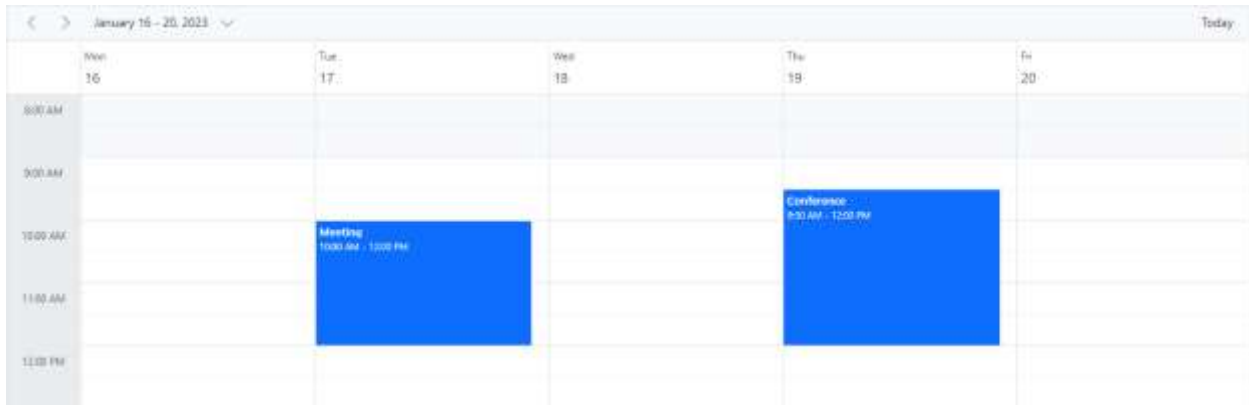
```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='WorkWeek' [workDays]="workWeekDays"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public workWeekDays: number[] = [2, 3, 5];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



The Week, Work week and Day views can display the all-day row appointments in a separate all-day row with an expand/collapse option to view it.

Month view

A Month view displays the entire days of a particular month and all its related appointments. You can navigate to a particular date in the day view by clicking on the appropriate date text on the month cells.

By default, when you try to create an appointment through Month view, it is considered as created for an entire day. You can explicitly change this behavior by unchecking the **All-day** option from editor window, so that it defaults to the start time duration as 9.00 AM and end time as 9.30 AM.

You can also have the **+ more** text indicator on each day cell of a Month view, clicking on which will allow you to view the hidden appointments of a day.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
```



```

    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
        <e-view option='Month' [showWeekNumber]="showWeekNumber"
[readonly]="isReadOnly"></e-view>
    </e-views>
</ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public showWeekNumber: boolean = true;
    public isReadOnly: boolean = true;
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  }

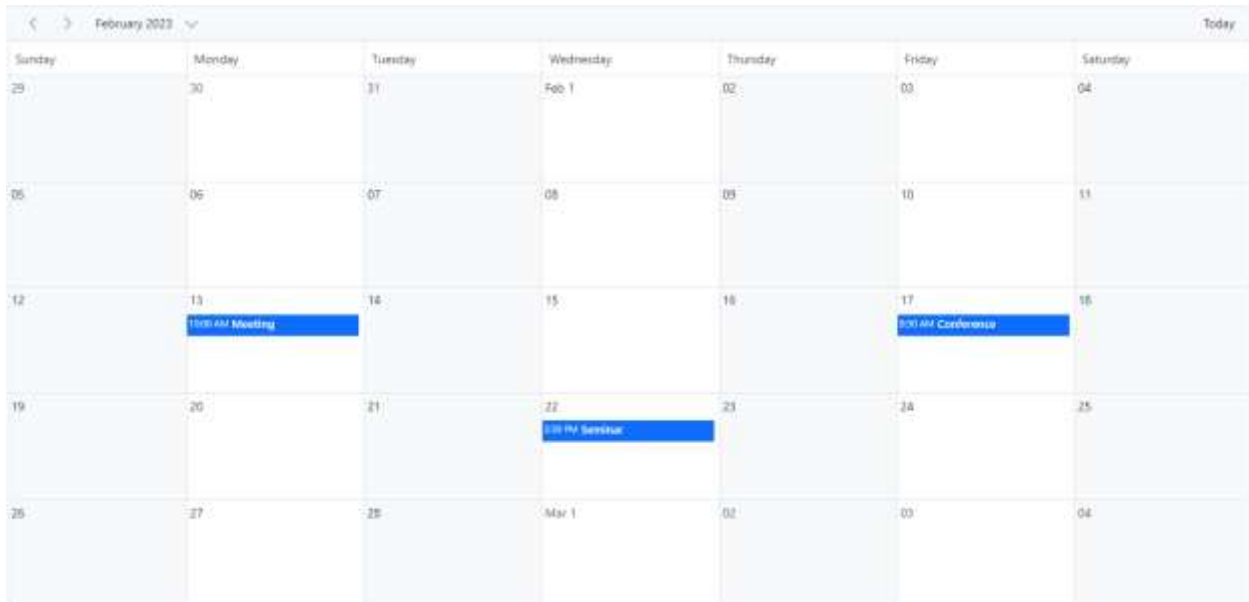
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Year view

A Year view displays all the days of a particular year with months and all its related appointments. You can navigate to a particular date in the day view by clicking on the appropriate date text on the year cells.

Year view is available in both the **Horizontal** and **Vertical** orientations. You can manage the orientation of year view through **views** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'

```

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, YearService } from '@syncfusion/ej2-angular-
schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    YearService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='Year' [showWeekNumber]="showWeekNumber"
[readonly]="isReadOnly"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public showWeekNumber: boolean = true;
  public isReadOnly: boolean = true;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The year view also has module support. In that, you can get all the months of a particular year in a calendar view format. In that calendar view, appointment contained dates are highlighted with dots placed under the individual date. When you click on the date, the event popup will be displayed and the events will be listed.

Agenda view

The Agenda view lists out the appointments in a grid-like view for the next 7 days by default from the current date. The count of the days can be changed using the API `agendaDaysCount`. It allows virtual scrolling of dates by enabling the `allowVirtualScrolling` property. Also, you can enable or disable the

display of days on Scheduler that has no appointments by setting true or false to the `hideEmptyAgendaDays` property.

The following code example depicts how to customize the display of events within Agenda view alone.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
import { EventSettingsModel, AgendaService } from '@syncfusion/ej2-angular-
schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='Agenda' [allowVirtualScrolling]="allowVirtualScroll">
        <ng-template #eventTemplate let-data>
          <div class="template-wrap">
            <div class="subject">{{data.Subject}}</div>
            <div class="time">{{getTimeString(data.StartTime)}} -
{{getTimeString(data.EndTime)}}</div>
          </div>
        </ng-template>
      </e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public allowVirtualScroll: boolean = true;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  private instance: Internationalization = new Internationalization();
  getTimeString(value: Date): string {
    return this.instance.formatDate(value, { skeleton: 'hm' });
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Schedule Height is mandatory to set in pixels for Agenda view.

Month Agenda view

A Month-Agenda view shows a month calendar, where clicking on a particular day will display the appointments present on that date below the calendar. The day with appointments are differentiated with a circular dot below the date of the calendar.

The following code example shows how to hide the weekend days on **MonthAgenda** view as well as the working days list is modified on Month Agenda view alone.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, MonthAgendaService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
```

```

    <e-view option='MonthAgenda' [showWeekend]="showWeekend"
[workDays]="workDays"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 14);
  public showWeekend: boolean = false;
  public workDays: number[] = [1, 2, 3];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Schedule Height is mandatory to set in pixels for Month Agenda view.

Timeline views – Day, Week, Work Week

Similar to the day view, it shows a single day with all its appointments where the time slots are displayed horizontally. By default, the cell height adjusts as per the height set to Scheduler. When the number of appointments exceeds the visible area of the cells, the **+ more** text indicator will be displayed at the bottom to denote the presence of few more appointments in that time range.

To make use of the timeline views (Timeline Day, Timeline Week and Timeline Work Week) on Scheduler, import and inject the module **TimelineViews** from the **ej2-schedule** package.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';

```

```
import { EventSettingsModel, TimelineViewsService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from '../datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='TimelineDay' startHour='10:00' endHour='15:30'></e-
view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Similar to the Week view, the timeline week view shows 7 days with its associated appointments with the time slots displayed horizontally.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService, TimeScaleModel } from
 '@syncfusion/ej2-angular-schedule';
import { scheduleData } from '../datasource';
@Component({
  imports: [
```

```

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService,
                TimelineViewsService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
        <e-views>
            <e-view option='TimelineWeek' timeScale="timeScaleOptions"></e-view>
        </e-views>
    </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public timeScaleOptions: TimeScaleModel = { enable: false };
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  }

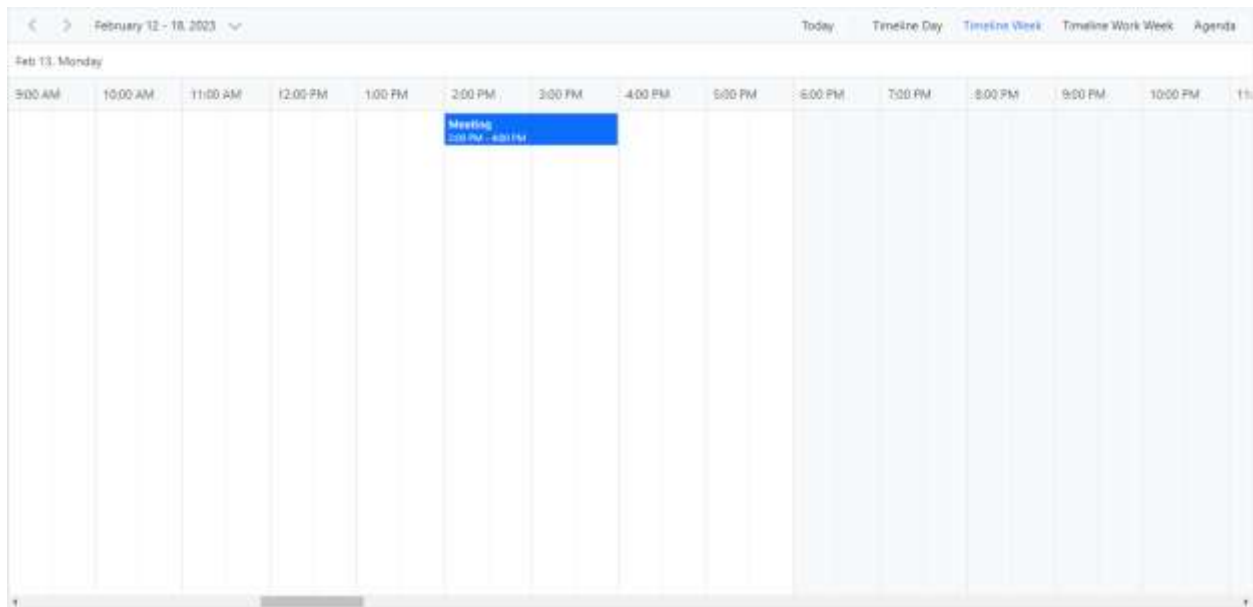
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



The following code example depicts how to display the timeline work week view on Scheduler,

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='TimelineWorkWeek' dateFormat='dd-MMM-yyyy'
[interval]="viewInterval" [workDays]="workweekDays"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 14);
  public viewInterval: number = 3;
  public workweekDays: number[] = [1, 3, 5];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Clicking on the dates in the date header bar of Timeline day, Timeline week and Timeline work week will allow you to navigate to the Agenda view.

Timeline Month view

A Timeline Month view displays the current month days along with its appointments. To make use of the timeline Month view on Scheduler, import and inject `TimelineMonth` module from the `ej2-schedule` package.

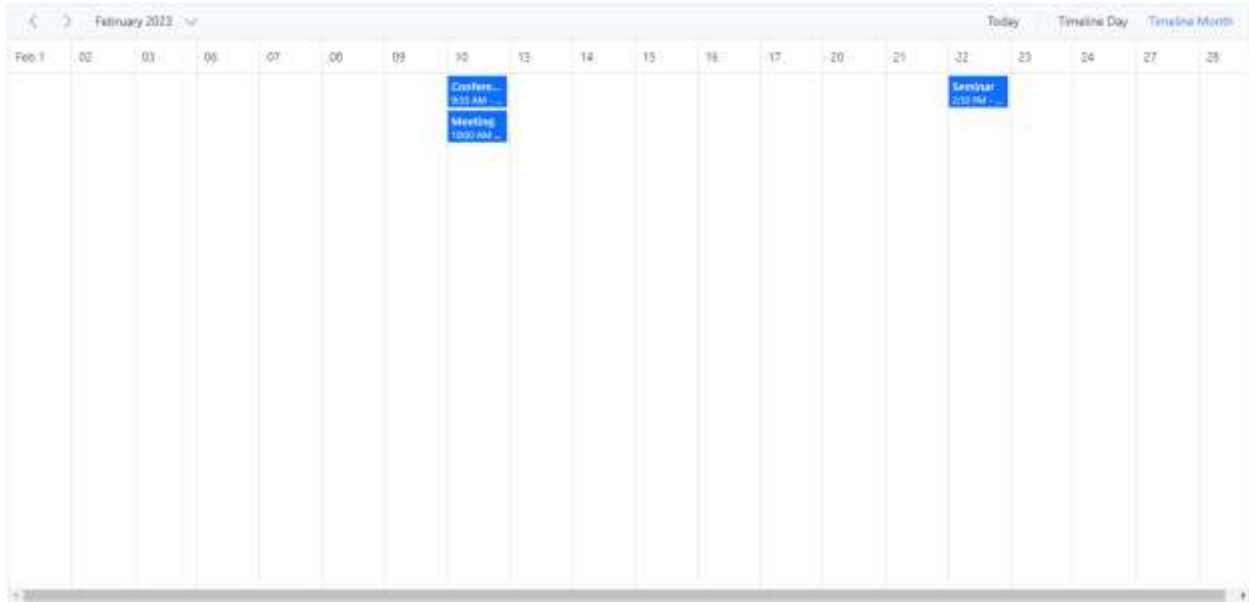
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from '../datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='TimelineMonth' dateFormat='dd-MMM-yyyy'
[showWeekend]="showWeekend"></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public showWeekend: boolean = false;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Clicking on the dates in the date header bar of Timeline month will allow you to navigate to the Timeline day view.

Timeline Year view

In Timeline Year view, each row depicts a single resource. Whereas in the vertical view, each resource is grouped parallelly as columns. Here, the resource grouping follows the tree-view like hierarchical grouping structure and can contain any level of child resources.

To make use of the timeline Year view on Scheduler, import and inject **TimelineYear** module from the **ej2-schedule** package.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineYearService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineYearService],
})
```

```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
    <e-views>
      <e-view option='TimelineYear' displayName='Horizontal Timeline Year'
isSelected=true></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

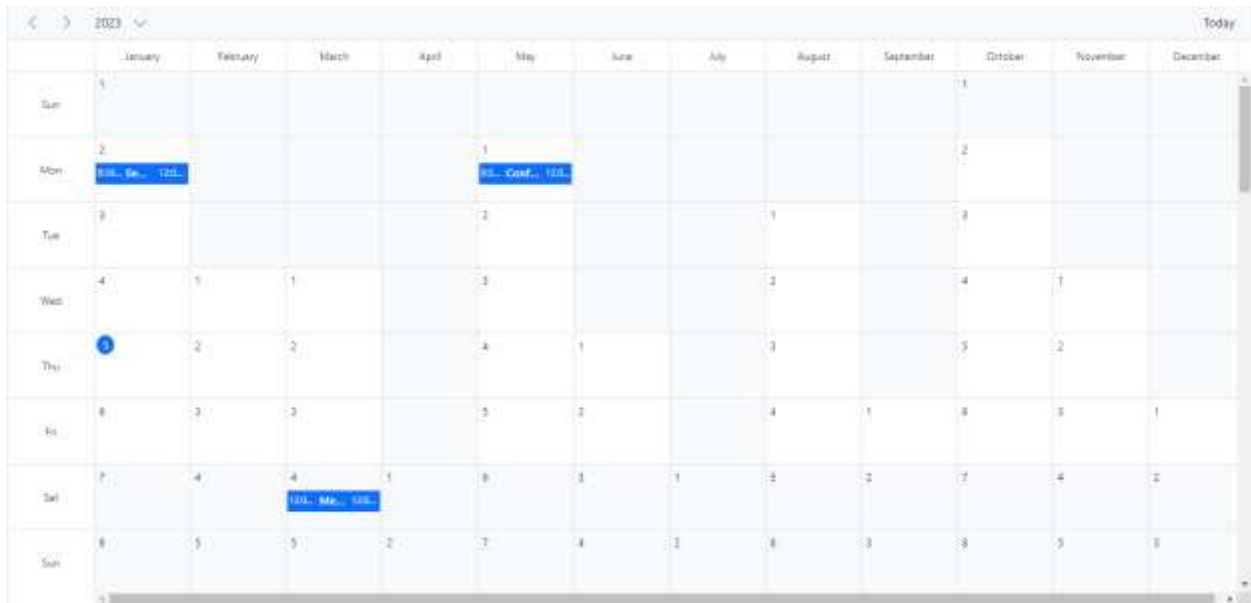
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Resource grouping

The following code example depicts how to group the multiple resources on Timeline Year view and its relevant events are displayed accordingly under those resources.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'

```

```

import { Component } from '@angular/core';
import { EventSettingsModel, TimelineYearService, GroupModel } from
 '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineYearService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [eventSettings]="eventSettings">
      <e-views>
        <e-view option='TimelineYear' displayName='Horizontal Timeline Year'
isSelected=true></e-view>
        <e-view option='TimelineYear' displayName='Vertical Timeline Year'
orientation='Vertical' [group]="groupSettings"></e-view>
      </e-views>
      <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
          [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
          textField="OwnerText" idField="Id" colorField="OwnerColor">
        </e-resource>
      </e-resources>
    </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ['TimelineYear'];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData
  };
  public groupSettings: GroupModel = { resources: ['Owners'] };
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';

```

```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Auto row height

Timeline Year view supports Auto row height. When the feature `rowAutoHeight` is enabled, the row height gets auto-adjusted based on the number of overlapping events occupied in the same time range. If you disable the Auto row height, you have the `+ more` text indicator on each day cell of a Timeline Year view, clicking on which will allow you to view the hidden appointments of a day.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineYearService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineYearService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [rowAutoHeight]='rowAutoHeight'
[eventSettings]='eventSettings'>
  <e-views>
    <e-view option='TimelineYear' displayName='Horizontal Timeline Year'
isSelected=true></e-view>
    <e-view option='TimelineYear' displayName='Vertical Timeline Year'
orientation='Vertical'></e-view>
  </e-views>
</ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 5, 17);
  public rowAutoHeight: boolean = true;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Extending view intervals

It is possible to customize the display of default number of days on different Scheduler view modes. For example, a day view can be extended to display 3 days by setting the `interval` option as 3 for the `Day` option within the `views` property as depicted in the following code example. In the same way, you can also display 2 weeks by setting interval 2 for the `Week` option.

You can provide the alternative display name for such customized views on the Scheduler header bar, by setting the appropriate `displayName` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' currentView="Day"
[selectedDate]="selectedDate" [eventSettings]="eventSettings" >
  <e-views>
    <e-view displayName='3 Days' option='Day'
[interval]="dayInterval"></e-view>
    <e-view displayName='2 Weeks' option='Week' [isSelected]="isSelected"
[interval]="weekInterval"></e-view>
  </e-views> </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public isSelected: Boolean = true;
  public dayInterval: number = 3;
  public weekInterval: number = 2;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
```

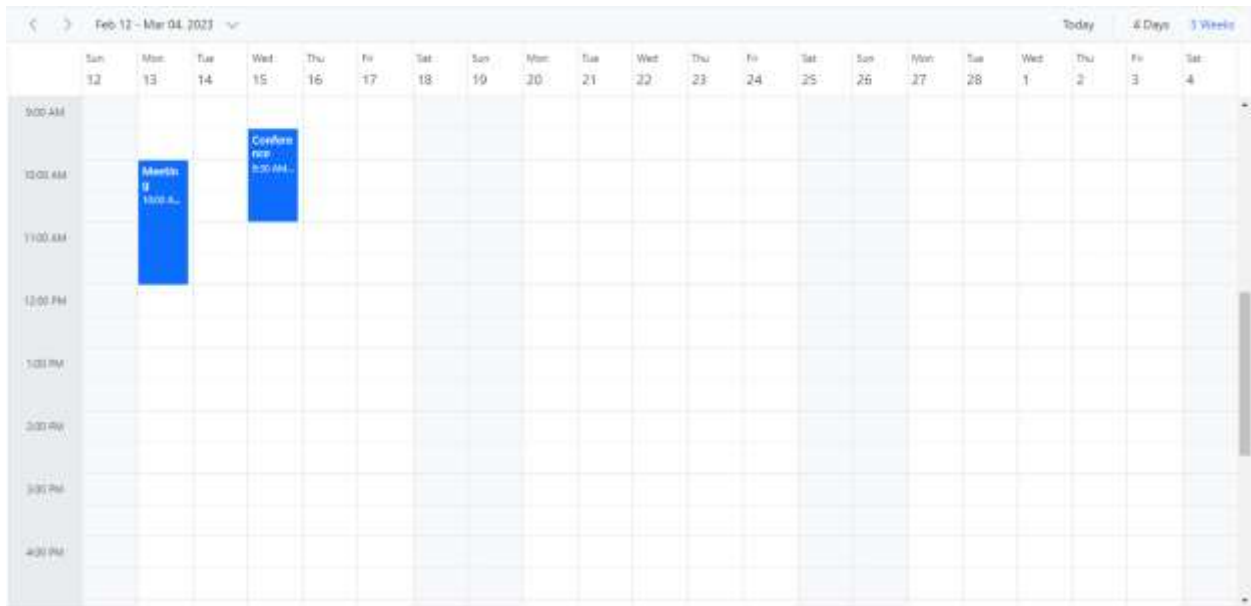
```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



The view intervals can be extended on all the Scheduler view modes except Agenda and Month-Agenda views.

See Also

- [How to restrict view navigation while clicking on dates](#)

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Calendar mode in Angular Schedule component

The Scheduler supports the following two types of calendar mode.

- Gregorian Calendar
- Islamic Calendar

Gregorian Calendar

The Scheduler by default displays the gregorian calendar dates which is the most widely used calendar in the world. The Gregorian calendar is a solar calendar which has 12 months with 28 to 31 days each. A year which is divisible by four is said to be a leap year in this calendar mode. A leap year usually has 366 days whereas the regular year has 365 days.

Islamic Calendar

The Islamic Calendar, also known as the Hijri or Muslim calendar, is a lunar calendar which has 12 months in a year with 354 or 355 days. Each month of this calendar denotes the birth of the new lunar cycle and therefore, each month can have 29 or 30 days depending on the visibility of the moon. Here, the odd-numbered months have 30 days and the even months have 29 days.

The current Islamic year is 1440 AH. Usually the Gregorian calendar runs from approximately 11 September 2018 to 30 August 2019 for this 1440 AH year.

The Scheduler has a property `calendarMode` which is used to switch between the gregorian and islamic calendar modes. By default, it is set to `Gregorian` and to use it with Islamic calendar dates, define the `calendarMode` of Scheduler to `Islamic`. The following example depicts, how to display the Islamic calendar dates on Scheduler.

To make use of Islamic calendar in Scheduler, import the `Calendar` and `Islamic` module from `ej2-calendars` package and also inject it using the `Calendar.Inject` method. Apart from this, it requires the following CLDR data to be loaded using `loadCldr` function.

- `numberingSystems.json`
- `ca-gregorian.json`
- `numbers.json`
- `timeZoneNames.json`
- `ca-islamic.json`

To know more information on, how to install the CLDR data, refer the [Internationalization](#) topic.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
import { Calendar, Islamic } from '@syncfusion/ej2-angular-calendars';
import {
  ScheduleComponent, DayService, WeekService, MonthService,
  MonthAgendaService,
  AgendaService, TimelineViewsService, TimelineMonthService,
  EventSettingsModel, WorkWeekService
} from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
import * as localeObj from './locale.json';
import * as numberingSystems from './numberingSystems.json';
import * as ca_gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import * as timeZoneNames from './timeZoneNames.json';
import * as ca_islamic from './ca-islamic.json';
Calendar.Inject(Islamic);
L10n.load(localeObj);
loadCldr(numberingSystems, ca_gregorian, numbers, timeZoneNames, ca_islamic);
@Component({
  imports: [
    ScheduleModule
```



```

    ],
    standalone: true,
    selector: 'app-root',
    providers: [DayService, WeekService, MonthService, AgendaService,
MonthAgendaService, TimelineViewsService, TimelineMonthService],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px' calendarMode='Islamic'
locale='ar' [enableRtl]="enableRtl" showQuickInfo="false"
[selectedDate]="selectedDate" [eventSettings]="eventSettings">
      <e-views>
        <e-view option='Day'></e-view>
        <e-view option='TimelineDay'></e-view>
        <e-view option='Week'></e-view>
        <e-view option='TimelineWeek'></e-view>
        <e-view option='Month'></e-view>
        <e-view option='TimelineMonth'></e-view>
        <e-view option='Agenda'></e-view>
        <e-view option='MonthAgenda'></e-view>
      </e-views>
    </ejs-schedule>`
  })
  export class AppComponent {
    public enableRtl: Boolean = true;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler exampleLink to the Video](#) to know how to present and manipulate data.

Resources in Angular Schedule component

Resources and grouping support allows the Scheduler to be shared by multiple resources. Also, the appointments of each resource are displayed under relevant resources. Each resource in the Scheduler is arranged in a column/row wise order, with individual spacing to display all its respective appointments on a single page. It also supports the multiple levels of grouping of resources, thus enabling the categorization of resources in a hierarchical structure and shows it either in expandable groups (Timeline views) or else vertical hierarchy one after the other (Calendar views).

It is also possible to assign one or more resources to the same appointment, by allowing multiple selection of resource options available in the event editor window.

The HTML5 JavaScript Scheduler groups the resources based on different criteria. It includes grouping appointments based on resources, grouping resources based on dates, and timeline scheduling. Also, the data for resources bind with Scheduler either as a local JSON collection or URL, retrieving data from remote data services.

Learn how to add appointments of multiple resources to the Angular Scheduler from this video:

Resource fields

The default options available within the `resources` collection are as follows,

Field name	Type	Description
-----	-----	-----
<code>field</code>	String	A value that binds to the resource field of event object.
<code>title</code>	String	It holds the title of the resource field to be displayed on the event editor window.
<code>name</code>	String	A unique resource name used for differentiating various resource objects while grouping.
<code>allowMultiple</code>	Boolean	When set to <code>true</code> , allows multiple selection of resource names, thus creating multiple instances of same appointment for the selected resources.
<code>dataSource</code>	Object	Assigns the resource <code>dataSource</code> , where data can be passed either as an array of JavaScript objects, or else can create an instance of DataManager in case of processing remote data and can be assigned to the <code>dataSource</code> property. With the remote data assigned to <code>dataSource</code> , check the available adaptors to customize the data processing.
<code>query</code>	Query	Defines the external query that will be executed along with the data processing.
<code>idField</code>	String	Binds the resource ID field name from the resources <code>dataSource</code> .
<code>expandedField</code>	String	Binds the <code>expandedField</code> name from the resources <code>dataSource</code> . It usually holds boolean value which decide whether the resource of timeline views is in collapse or expand state on initial load.
<code>textField</code>	String	Binds the text field name from the resources <code>dataSource</code> . It usually holds the resource names.
<code>groupIDField</code>	String	Binds the group ID field name from the resource <code>dataSource</code> . It usually holds the value of resource IDs of parent level resources.
<code>colorField</code>	String	Binds the color field name from the resource <code>dataSource</code> . The color value mapped in this field will be applied to the events of resources.
<code>startHourField</code>	String	Binds the start hour field name from the resource <code>dataSource</code> . It allows to provide different work start hour for the resources.
<code>endHourField</code>	String	Binds the end hour field name from the resource <code>dataSource</code> . It allows to provide different work end hour for the resources.
<code>workDaysField</code>	String	Binds the work days field name from the resources <code>dataSource</code> . It allows to provide different working days collection for the resources.
<code>cssClassField</code>	String	Binds the custom CSS class field name from the resources <code>dataSource</code> . It maps the CSS class written for the specific resources and applies it to the events of those resources.

Resource data binding

The data for resources can bind with Scheduler either as a local JSON collection or a service URL, retrieving resource data from remote data services.

Using local JSON data

The following code example depicts how to bind the local JSON data to the `dataSource` of `resources` collection.

```
`typescript
import { Component } from '@angular/core';
import {
  WeekService, MonthService, AgendaService, TimelineViewsService, TimelineMonthService,
  EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource.ts';
@Component({
  selector: "app-root",
  providers: [WeekService, MonthService, AgendaService, TimelineViewsService, TimelineMonthService],
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate" [views]="views"
    [eventSettings]="eventSettings">
      <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
        textField="OwnerText" idField="Id" colorField="OwnerColor">
      </e-resource>
    </e-resources>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ['Week', 'Month', 'TimelineWeek', 'TimelineMonth', 'Agenda'];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData
  };
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
```

```
{ OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
{ OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
];
}
```

Using remote service URL

The following code example depicts how to bind the remote data for resources `dataSource`.

```
`typescript
import { Component } from '@angular/core';
import {
  WeekService, MonthService, AgendaService, TimelineViewsService, TimelineMonthService,
  EventSettingsModel
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource.ts';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: "app-root",
  providers: [WeekService, MonthService, AgendaService, TimelineViewsService, TimelineMonthService],
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate" [views]="views"
    [eventSettings]="eventSettings">
      <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="resourceDataManager" [allowMultiple]="allowMultipleOwner"
        textField="OwnerText" idField="Id" colorField="OwnerColor">
      </e-resource>
    </e-resources>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ['Week', 'Month', 'TimelineWeek', 'TimelineMonth', 'Agenda'];
  public resourceDataManager: DataManager = new DataManager({
```

```

url: 'Home/GetResourceData',
adaptor: new UrlAdaptor,
crossDomain: true
});

public eventSettings: EventSettingsModel = {
dataSource: resourceData
};

public allowMultipleOwner: Boolean = true;
}
`

```

Scheduler with multiple resources

It is possible to display the Scheduler in default mode without visually showcasing all the resources in it, but allowing to assign the required resources to the appointments through the event editor resource options.

The appointments belonging to the different resources will be displayed altogether on the default Scheduler, which will be differentiated based on the resource color assigned in the **resources** (depicting to which resource that particular appointment belongs) collection.

Example: To display default Scheduler with multiple resource options in the event editor, ignore the group option and simply define the **resources** property with all its internal options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import {
    WeekService, MonthService, AgendaService, TimelineViewsService,
    TimelineMonthService, EventSettingsModel, DayService,
    WorkWeekService,
    MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from '../datasource';
@Component({
imports: [
    ScheduleModule,
    RadioButtonModule
],
standalone: true,
selector: "app-root",
providers: [WeekService, MonthService, AgendaService,
    TimelineViewsService, TimelineMonthService, DayService,
    WorkWeekService,
    MonthAgendaService],
// specifies the template string for the Schedule component

```

```

    template: `
      <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
  [eventSettings]="eventSettings">
    <e-resources>
      <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
        textField="OwnerText" idField="Id" colorField="OwnerColor">
      </e-resource>
    </e-resources>
  </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 1);
    public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth', 'Agenda'];
    public eventSettings: EventSettingsModel = {
      dataSource: resourceData
    };
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
      { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
      { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
      { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
    ];
  }
}

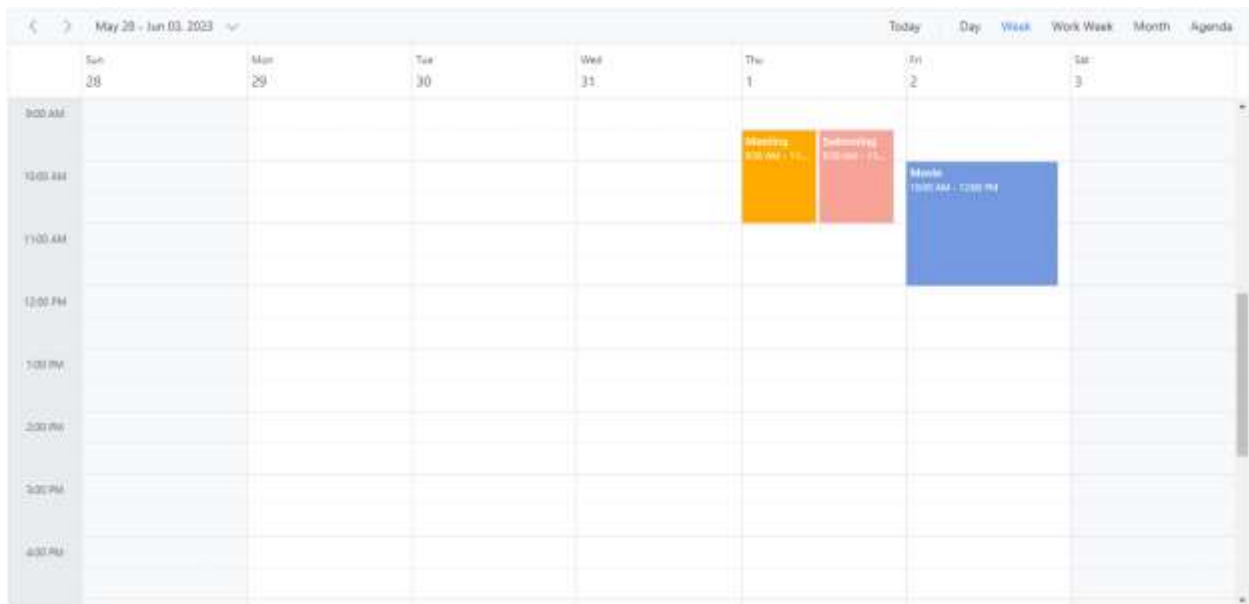
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Setting `allowMultiple` to [Link to the Video](#) in the above code example allows you to select multiple resources from the event editor and also creates multiple copies of the same appointment in the Scheduler for each resources while rendering.

Resource grouping

Resource grouping support allows the Scheduler to group the resources in a hierarchical structure both as an expandable groups (Timeline views) and as vertical hierarchy displaying resources one after the other (Resources view).

Scheduler supports both single and multiple levels of resource grouping that can be customized both in timeline and vertical Scheduler views.

Explore the advanced options available with the multiple resources and grouping concepts of Angular Scheduler by watching this video:

Vertical resource view

The following code example displays how the multiple resources are grouped and its events are portrayed in the default calendar views.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, EventSettingsModel, GroupModel } from
 '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { resourceConferenceData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [eventSettings]="eventSettings" [group]='group'>
    <e-resources>
      <e-resource field="ProjectId" title="Choose Project" name="Projects"
        [dataSource]="projectDataSource"
        textField="text" idField="id" colorField="color">
      </e-resource>
      <e-resource field="TaskId" title="Category" name="Categories"
```

```

        [dataSource]="categoryDataSource"
    [allowMultiple]="allowMultipleCategory"
        textField="text" idField="id" colorField="color">
    </e-resource>
</e-resources>
</ejs-schedule>`
    })
    export class AppComponent {
        public selectedDate: Date = new Date(2018, 3, 1);
        public views: Array<string> = ['Week', 'Month', 'Agenda'];
        public eventSettings: EventSettingsModel = {
            dataSource: resourceConferenceData
        };
        public group: GroupModel = {
            byGroupID: false,
            resources: ['Projects', 'Categories']
        };
        public projectDataSource: Object[] = [
            { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
            { text: 'PROJECT 2', id: 2, color: '#56ca85' },
            { text: 'PROJECT 3', id: 3, color: '#df5286' }
        ];
        public allowMultipleCategory: Boolean = true;
        public categoryDataSource: Object[] = [
            { text: 'Development', id: 1, color: '#df5286' },
            { text: 'Testing', id: 2, color: '#7fa900' }
        ];
    }

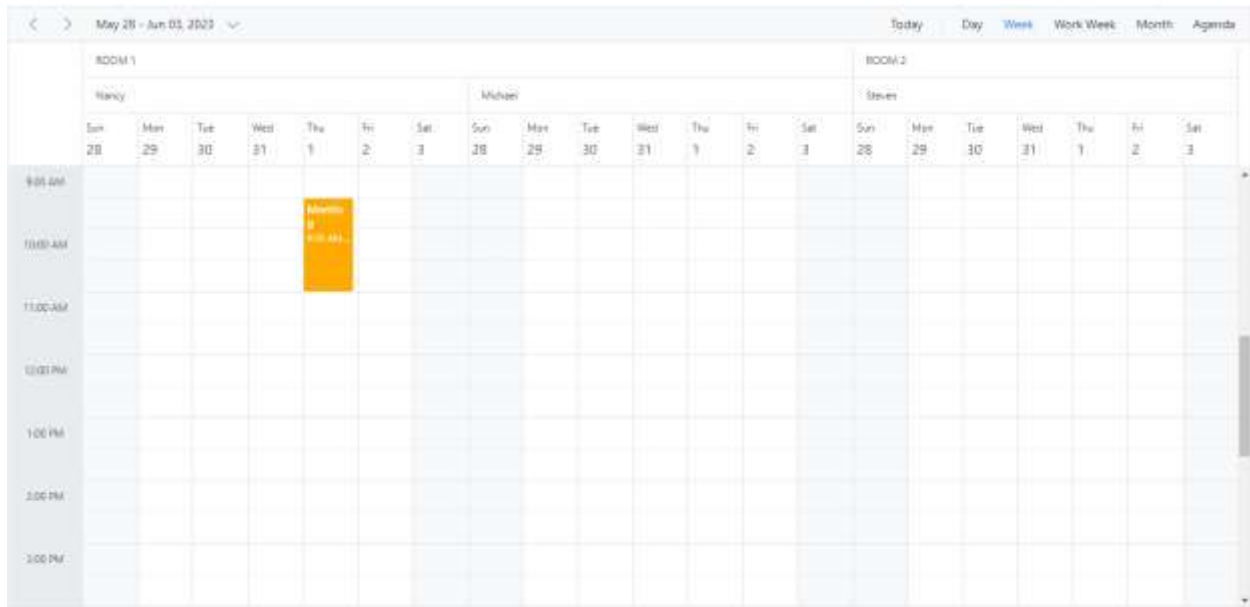
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Timeline resource view

The following code example depicts how to group the multiple resources on Timeline Scheduler views and its relevant events are displayed accordingly under those resources.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
    DayService, WeekService, WorkWeekService, MonthService,
    MonthAgendaService, TimelineViewsService, TimelineMonthService,
    AgendaService, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService, TimelineMonthService],
    standalone: true,
    selector: "app-root",
    // specifies the template string for the Schedule component
    template: `
        <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
        [views]="views"
    `
})
```

```

    [currentView]='currentView' [eventSettings]="eventSettings"
    [group]='group'>
      <e-resources>
        <e-resource field="RoomId" title="Room" name="Rooms"
          [dataSource]="roomDataSource"
          textField="RoomText" idField="Id" colorField="RoomColor">
        </e-resource>
        <e-resource field="OwnerId" title="Owner" name="Owners"
          [dataSource]="ownerDataSource"
[allowMultiple]="allowMultipleCategory"
          textField='OwnerText' idField='Id' groupIDField='OwnerGroupId'
colorField='OwnerColor'>
        </e-resource>
      </e-resources>
    </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 4);
    public currentView: string = 'TimelineWeek';
    public views: Array<string> = ['TimelineWeek', 'TimelineMonth',
'Agenda'];
    public eventSettings: EventSettingsModel = {
      dataSource: resourceData
    };
    public group: GroupModel = {
      resources: ['Rooms', 'Owners']
    };
    public roomDataSource: Object[] = [
      { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
      { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
    ];
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
      { OwnerText: 'Nancy', Id: 1, OwnerGroupId: 1, OwnerColor: '#ffaa00'
},
      { OwnerText: 'Steven', Id: 2, OwnerGroupId: 2, OwnerColor: '#f8a398'
},
      { OwnerText: 'Michael', Id: 3, OwnerGroupId: 1, OwnerColor: '#7499e1'
}
    ];
    allowMultipleCategory: any;
  }
}

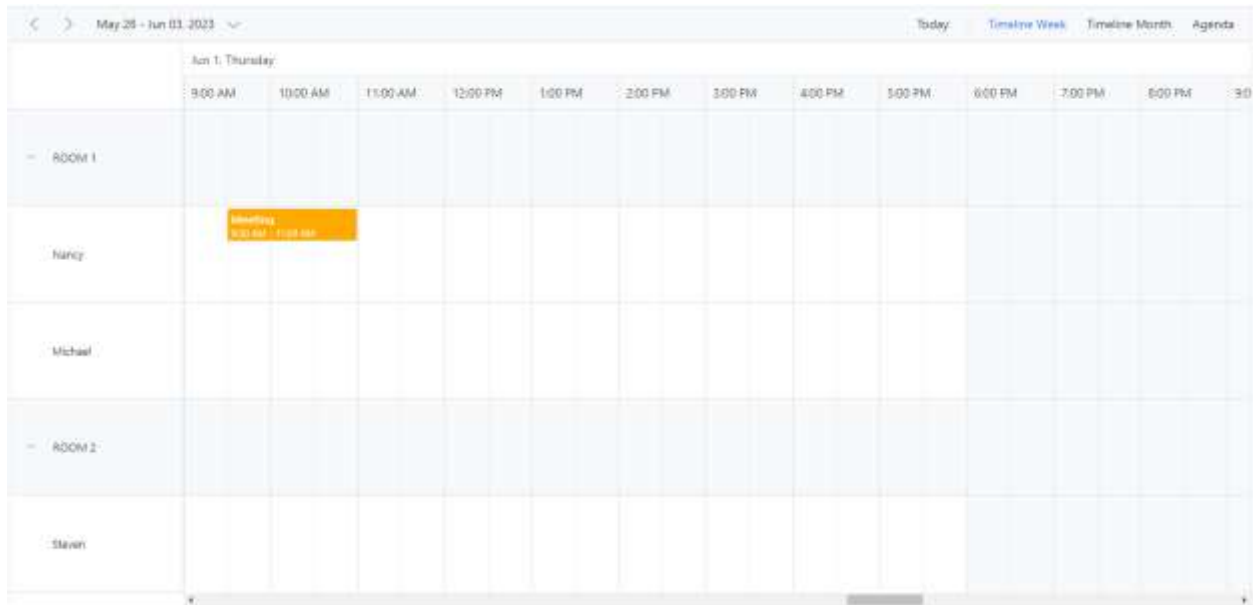
```

MAIN.TS

```

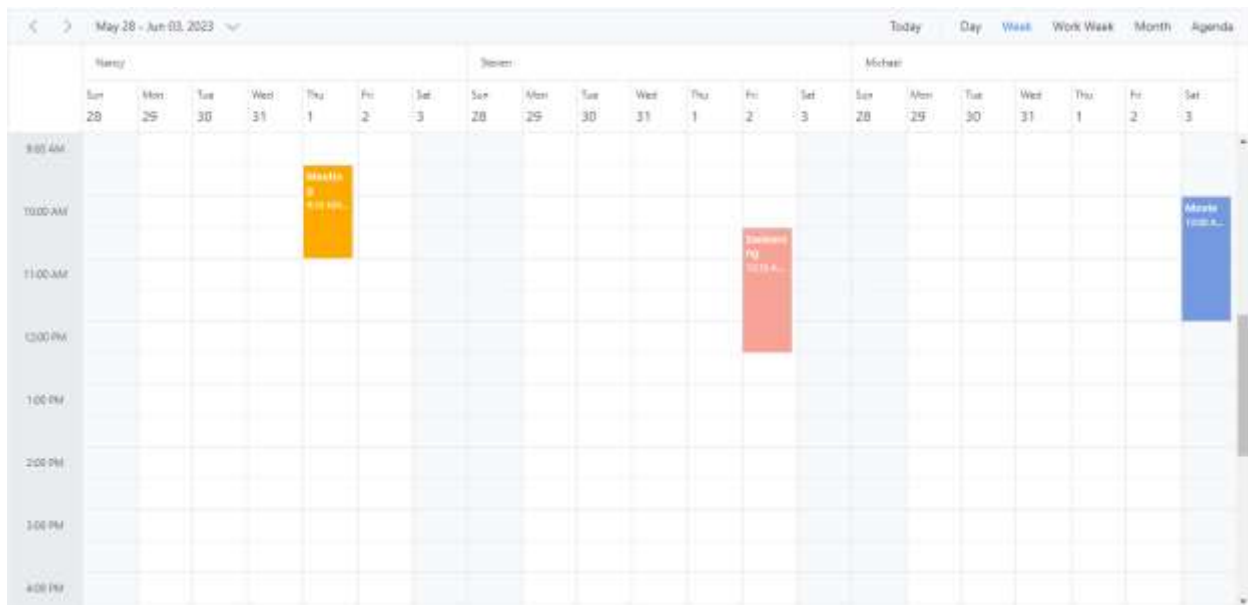
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Grouping single-level resources

This kind of grouping allows the Scheduler to display all the resources at a single level simultaneously. The appointments mapped under resources will be displayed with the colors as per the `colorField` defined on the resources collection.



Example: To display the Scheduler with single level resource grouping,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
```

```

    WorkWeekService, WeekService, MonthService, AgendaService,
    TimelineViewsService, TimelineMonthService, EventSettingsModel, GroupModel,
    DayService, MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
    [views]="views"
      [eventSettings]="eventSettings" [group]='group'>
      <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
          [dataSource]="ownerDataSource"
        [allowMultiple]="allowMultipleCategory"
          textField='OwnerText' idField='Id' colorField='OwnerColor'>
        </e-resource>
      </e-resources>
    </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 1);
    public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth', 'Agenda'];
    public eventSettings: EventSettingsModel = {
      dataSource: resourceData
    };
    public group: GroupModel = {
      resources: ['Owners']
    };
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
      { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
      { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
      { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
    ];
    allowMultipleCategory: any;
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';

```

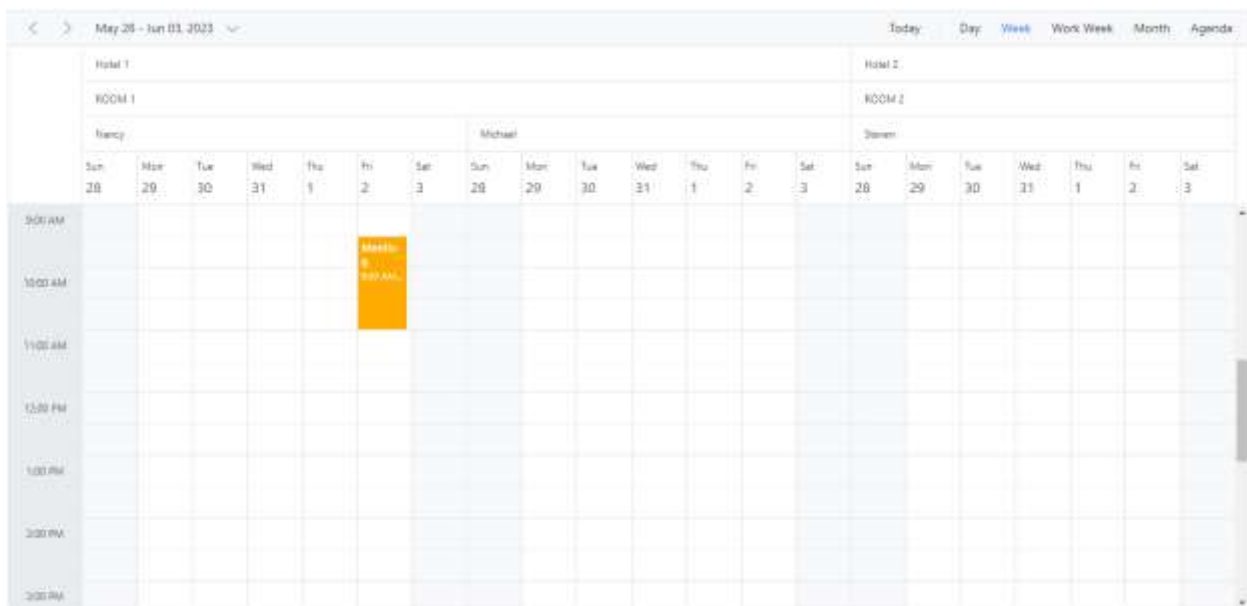
```
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

![Grouping single-level resources](images/schedule-singlelevel.png)

The **name** field defined in the **resources** collection namely **Owners** will be mapped within the **group** property, in order to enable the grouping option with those resource levels on the Scheduler.

Grouping multi-level resources

It is possible to group the resources of Scheduler in multiple levels, by mapping the child resources to each parent resource. In the following example, there are 2 levels of resources, on which the second level resources are defined with **groupId** mapping to the first level resource's ID so as to establish the parent-child relationship between them.



Example: To display the Scheduler with multiple level resource grouping options,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SchedulerModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
    WeekService, MonthService, AgendaService, TimelineViewsService,
    TimelineMonthService, DayService, MonthAgendaService, WorkWeekService,
    EventSettingsModel, GroupModel } from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
    imports: [
        SchedulerModule,
        TimePickerModule
    ],
    standalone: true,
```

```

    selector: "app-root",
    providers: [WeekService, WorkWeekService, MonthService, AgendaService,
TimelineViewsService, TimelineMonthService ],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width="100%" height="550px"
[selectedDate]="selectedDate"
    [views]="views" [currentView]="currentView"
[eventSettings]="eventSettings" [group]="group">
      <e-resources>
        <e-resource field="RoomId" title="Room" name="Rooms"
[dataSource]="roomDataSource"
          textField="RoomText" idField="Id" colorField="RoomColor">
        </e-resource>
        <e-resource field="OwnerId" title="Owner" name="Owners"
[dataSource]="ownerDataSource"
[allowMultiple]="allowMultipleCategory" textField="OwnerText"
          idField="Id" groupIDField="OwnerGroupId" colorField="OwnerColor">
        </e-resource>
      </e-resources>
    </ejs-schedule>
  `;
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ["Week", "Month", "TimelineWeek",
"TimelineMonth", "Agenda"];
  public eventSettings: EventSettingsModel = { dataSource: resourceData };
  public group: GroupModel = {
    resources: ["Rooms", "Owners"]
  };
  public roomDataSource: Object[] = [
    { RoomText: "ROOM 1", Id: 1, RoomColor: "#cb6bb2" },
    { RoomText: "ROOM 2", Id: 2, RoomColor: "#56ca85" }
  ];
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { OwnerText: "Nancy", Id: 1, OwnerGroupId: 1, OwnerColor: "#ffaa00" },
    { OwnerText: "Steven", Id: 2, OwnerGroupId: 2, OwnerColor: "#f8a398" },
    { OwnerText: "Michael", Id: 3, OwnerGroupId: 1, OwnerColor: "#7499e1" }
  ];
  currentView: any;
  allowMultipleCategory: any;
}

```

MAIN.TS

```

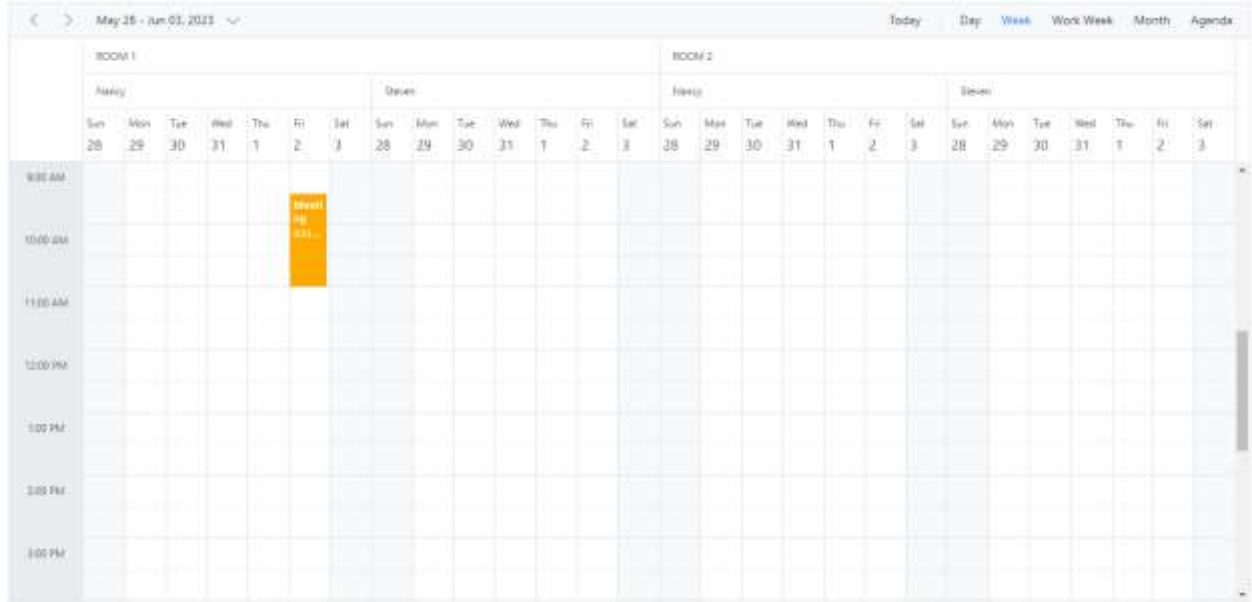
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Grouping multi-level resources](images/schedule-multiplelevel.png)

One-to-One grouping

In multi-level grouping, Scheduler usually groups the resources on the child level based on the **GroupID** that maps with the **Id** field of parent level resources (as **byGroupID** set to true by default). There are also option which allows you to group all the child resource(s) against each of its parent resource(s). To enable this kind of grouping, set **false** to the **byGroupID** option within the



group property. In the following code example, there are two levels of resources, on which all the 3 resources at the child level is mapped one to one with each resource on the first level.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
  WorkWeekService, DayService, WeekService, MonthAgendaService, MonthService,
  AgendaService, TimelineViewsService, TimelineMonthService,
  EventSettingsModel, GroupModel
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: 'app-root',
```

```

// specifies the template string for the Schedule component
template: `
  <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [currentView]="currentView" [eventSettings]="eventSettings"
[group]="group">
    <e-resources>
      <e-resource field="RoomId" title="Room" name="Rooms"
[dataSource]="roomDataSource"
        textField="RoomText" idField="Id" colorField="RoomColor">
      </e-resource>
      <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
        textField="OwnerText" idField="Id" colorField="OwnerColor">
      </e-resource>
    </e-resources>
  </ejs-schedule>
`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ["Week", "Month", "TimelineWeek",
"TimelineMonth", "Agenda"];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData
  };
  public group: GroupModel = {
    byGroupID: false,
    resources: ['Rooms', 'Owners']
  };
  public roomDataSource: Object[] = [
    { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
    { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
  ];
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
  ];
  currentView: any;
}

```

MAIN.TS

```

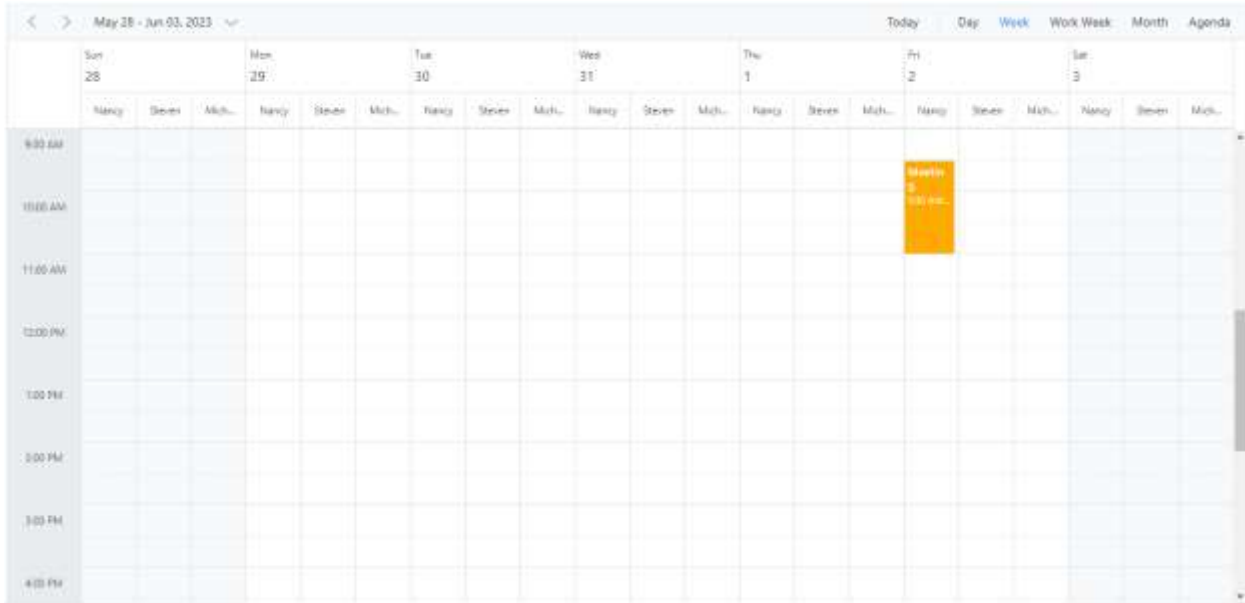
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Grouping in Schedule](images/schedule-grouping.png)

Grouping resources by date

It groups the number of resources under each date and is applicable only on the calendar views such as Day, Week, Work Week, Month, Agenda and Month-Agenda. To enable such grouping, set `byDate` option to `true` within the `group` property.



Example: To display the Scheduler with resources grouped by date,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, GroupModel
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
    [views]="views"
  `
})
```

```

    [eventSettings]="eventSettings" [group]='group'>
    <e-resources>
      <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
        textField='text' idField='id' colorField='color'>
      </e-resource>
    </e-resources>
  </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 1);
    public views: Array<string> = ['Week', 'Month', 'Agenda'];
    public eventSettings: EventSettingsModel = {
      dataSource: resourceData
    };
    public group: GroupModel = {
      byDate: true,
      resources: ['Owners']
    };
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
      { text: 'Alice', id: 1, color: '#1aaa55' },
      { text: 'Smith', id: 2, color: '#7fa900' }
    ];
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Grouping Resources by Date in Schedule](images/schedule-resource-by-date.png)

This kind of grouping by date is not applicable on any of the timeline views.

Customizing parent resource cells

In timeline view work cells of parent resource can be customized by checking the `elementType` as `resourceGroupCells` in the event `renderCell`. In the following code example, background color of the work hours has been changed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
  AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { TimelineViewsService, TimelineMonthService, EventSettingsModel,
  GroupModel, RenderCellEventArgs } from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({

```

```

imports: [
    ScheduleModule,
    ButtonModule
],
providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService,
            TimelineViewsService, TimelineMonthService],
standalone: true,
selector: "app-root",
// specifies the template string for the Schedule component
template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views" currentView='TimelineWeek'
    [eventSettings]="eventSettings" [group]='group'
(renderCell)="onRenderCell($event)">
    <e-resources>
        <e-resource field="RoomId" title="Room" name="Rooms"
            [dataSource]="roomDataSource" [allowMultiple]="allowMultipleRoom"
            textField='text' idField='id' colorField='color'></e-resource>
        <e-resource field="OwnerId" title="Owner" name="Owners"
            [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
            textField='text' idField='id' colorField='color'
groupIDField='groupId'></e-resource>
    </e-resources>
    </ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 4);
    public views: Array<string> = ['TimelineDay', 'TimelineWeek',
'TimelineMonth'];
    public eventSettings: EventSettingsModel = { dataSource: resourceData };
    public group: GroupModel = { resources: ['Rooms', 'Owners'] };
    public allowMultipleRoom: Boolean = false;
    public allowMultipleOwner: Boolean = true;
    public roomDataSource: Object[] = [
        { text: 'ROOM 1', id: 1, color: '#cb6bb2' },
        { text: 'ROOM 2', id: 2, color: '#56ca85' }
    ];
    public ownerDataSource: Object[] = [
        { text: 'Nancy', id: 1, groupId: 1, color: '#ffaa00' },
        { text: 'Steven', id: 2, groupId: 2, color: '#f8a398' },
        { text: 'Michael', id: 3, groupId: 1, color: '#7499e1' }
    ];
    public onRenderCell(args: RenderCellEventArgs): void {
        if (args.elementType == 'resourceGroupCells' &&
args.element.className.indexOf('e-work-hours') > 0) {
            (args.element as any).style.background = "#FAFAE3";
        }
    }
}
}

```

MAIN.TS

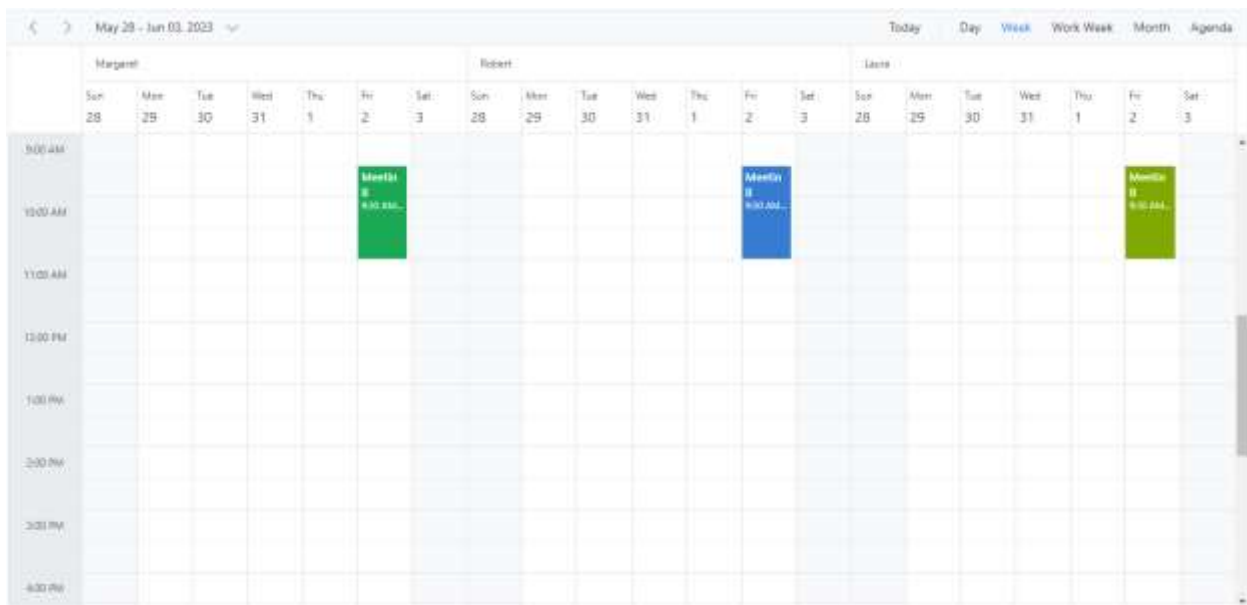
```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Working with shared events

Multiple resources can share the same events, thus allowing the CRUD action made on it to reflect on all other shared instances simultaneously. To enable such option, set `allowGroupEdit` option to `true` within the `group` property. With this property enabled, a single appointment

object will be maintained within the appointment collection, even if it is shared by more than one resource – whereas the resource fields of such appointment object will be in array which hold the IDs of the multiple resources.

Any actions such as create, edit or delete held on any one of the shared event instances, will be reflected on all other related instances visible on the UI.



Example: To edit all the resource events simultaneously,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
    DayService, WorkWeekService, WeekService, MonthAgendaService,
    MonthService, AgendaService, TimelineViewsService, TimelineMonthService,
    ResizeService, EventSettingsModel, GroupModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
    imports: [
```

```

        ScheduleModule,
        TimePickerModule
    ],
    standalone: true,
    selector: "app-root",
    providers: [WorkWeekService, WeekService, MonthService, AgendaService,
TimelineViewsService, TimelineMonthService, ResizeService, WorkWeekService,
MonthAgendaService, DayService],
    // specifies the template string for the Schedule component
    template: `
        <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
        [eventSettings]="eventSettings" [group]='group'>
            <e-resources>
                <e-resource field="ConferenceId" title="Conference"
name="Conferences"
                [dataSource]="conferenceDataSource"
[allowMultiple]="allowMultipleCategory"
                textField='Text' idField='Id' colorField='Color'>
            </e-resource>
        </e-resources>
    </ejs-schedule>`
    })
    export class AppComponent {
        public selectedDate: Date = new Date(2018, 5, 5);
        public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth', 'Agenda'];
        public eventSettings: EventSettingsModel = {
            dataSource: [
                {
                    Id: 1,
                    Subject: 'Burning Man',
                    StartTime: new Date(2018, 5, 1, 15, 0),
                    EndTime: new Date(2018, 5, 1, 17, 0),
                    ConferenceId: [1, 2, 3]
                }, {
                    Id: 2,
                    Subject: 'Data-Driven Economy',
                    StartTime: new Date(2018, 5, 2, 12, 0),
                    EndTime: new Date(2018, 5, 2, 14, 0),
                    ConferenceId: [1, 2]
                }, {
                    Id: 3,
                    Subject: 'Techweek',
                    StartTime: new Date(2018, 5, 2, 15, 0),
                    EndTime: new Date(2018, 5, 2, 17, 0),
                    ConferenceId: [2, 3]
                }, {
                    Id: 4,
                    Subject: 'Content Marketing World',
                    StartTime: new Date(2018, 5, 2, 18, 0),
                    EndTime: new Date(2018, 5, 2, 20, 0),
                    ConferenceId: [1, 3]
                }, {
                    Id: 5,
                    Subject: 'B2B Marketing Forum',
                    StartTime: new Date(2018, 5, 3, 10, 0),

```

```

        EndTime: new Date(2018, 5, 3, 12, 0),
        ConferenceId: [1, 2, 3]
    }
    ];
    public group: GroupModel = {
        allowGroupEdit: true,
        resources: ['Conferences']
    };
    public allowMultipleCategory: Boolean = true;
    public conferenceDataSource: Object[] = [
        { Text: 'Margaret', Id: 1, Color: '#1aaa55' },
        { Text: 'Robert', Id: 2, Color: '#357cd2' },
        { Text: 'Laura', Id: 3, Color: '#7fa900' }
    ];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Grouping Resources as shared events](images/schedule-sharedevents.png)

Simple resource header customization

It is possible to customize the resource header cells using built-in template option and change the look and appearance of it in both the vertical and timeline view modes. All the resource related fields and other information can be accessed within the resource header template option.



Example: To customize the resource header and display it along with designation field, refer the below code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel, GroupModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { doctorData } from '../datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [eventSettings]="eventSettings" [group]='group'>
      <e-resources>
        <e-resource field="DoctorId" title="Doctor Name" name="Doctors"
          [dataSource]="doctorDataSource"
          textField='text' idField='id' colorField='color'
designationField='designation'>
          </e-resource>
        </e-resources>
        <ng-template #resourceHeaderTemplate let-data>
          <div class='template-wrap'>
            <div class="resource-details">
              <div class="resource-
name">{{data.resourceData.text}}</div>
              <div class="resource-
designation">{{data.resourceData.designation}}</div>
            </div>
          </div>
        </ng-template>
      </ejs-schedule>`
  })
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 5);
  public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth', 'Agenda'];
  public eventSettings: EventSettingsModel = {
    dataSource: doctorData
  };
  public group: GroupModel = {
    resources: ['Doctors']
  }
}

```

```

    };
    public allowMultipleDoctors: Boolean = true;
    public doctorDataSource: Object[] = [
      { text: 'Will Smith', id: 1, color: '#ea7a57', designation:
'Cardiologist' },
      { text: 'Alice', id: 2, color: '#7fa900', designation: 'Neurologist'
},
      { text: 'Robson', id: 3, color: '#7fa900', designation: 'Orthopedic
Surgeon' }
    ];
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![[Customizing Resources Header in Desktop]](images/schedule-custom-resource-header.png)

To customize the resource header in compact mode properly make use of the class

June 11 - 17, 2023

Today

Timeline Week

Timeline Month

Jun 16, Friday

6:00 AM

7:00 AM

8:00 AM

9:00 AM

10:00 AM

11:00 AM

12:00 PM

1:00 PM

2:00 PM

3:00 PM

4:00 PM

Jenny

20

Conference 10

Twenty

7

Cabin 5

Hattie

5

Cabin 2

Phoenix

15

Conference 12

Rick Roll

20

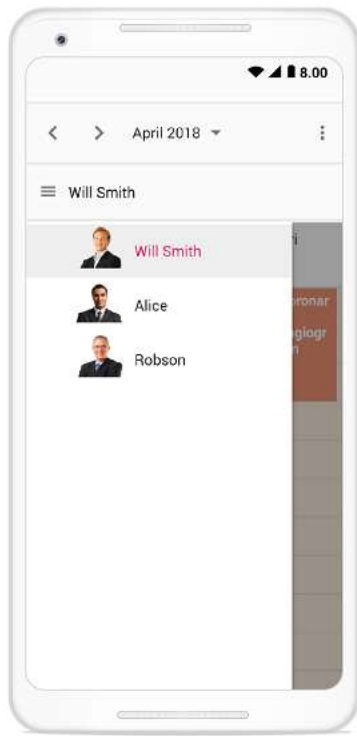
Conference 7

Rainbow

8

Cabin 3

4



e-device as in the code

example.

![Resource header template in compact mode](./images/header-template.png)

Customizing resource header with multiple columns

It is possible to customize the resource headers to display with multiple columns such as Room, Type and Capacity. The following code example depicts the way to achieve it and is applicable only on timeline views.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import {
    DayService, WeekService, WorkWeekService, MonthService, AgendaService,
    MonthAgendaService, TimelineViewsService, TimelineMonthService,
    EventSettingsModel, GroupModel, RenderCellEventArgs
} from '@syncfusion/ej2-angular-schedule';
import { roomData } from '../datasource';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
```

```

        AgendaService,
        MonthAgendaService,
        TimelineViewsService, TimelineMonthService],
standalone: true,
selector: "app-root",
// specifies the template string for the Schedule component
template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [eventSettings]="eventSettings" [group]='group'
(renderCell)='onRenderCell($event)' '>
    <e-resources>
        <e-resource field="RoomId" title="Room Type" name="MeetingRoom"
        [dataSource]="roomDataSource" [allowMultiple]='allowMultipleRoom'
        textField='text' idField='id' colorField='color'>
        </e-resource>
    </e-resources>
    <ng-template #resourceHeaderTemplate let-data>
        <div class='template-wrap'>
            <div class="room-name">{{data.resourceData.text}}</div>
            <div class="room-type">{{data.resourceData.type}}</div>
            <div class="room-
capacity">{{data.resourceData.capacity}}</div>
        </div>
    </ng-template>
    </ejs-schedule>`
))
export class AppComponent {
    public selectedDate: Date = new Date(2018, 7, 1);
    public views: Array<string> = ['TimelineWeek', 'TimelineMonth'];
    public eventSettings: EventSettingsModel = {
        dataSource: roomData
    };
    public group: GroupModel = {
        resources: ['MeetingRoom']
    };
    public allowMultipleRoom: Boolean = true;
    public roomDataSource: Object[] = [
        { text: 'Jammy', id: 1, color: '#ea7a57', capacity: 20, type:
'Conference' },
        { text: 'Tweety', id: 2, color: '#7fa900', capacity: 7, type: 'Cabin'
},
        { text: 'Nestle', id: 3, color: '#5978ee', capacity: 5, type: 'Cabin'
},
        { text: 'Phoenix', id: 4, color: '#fec200', capacity: 15, type:
'Conference' },
        { text: 'Mission', id: 5, color: '#df5286', capacity: 25, type:
'Conference' },
        { text: 'Hangout', id: 6, color: '#00bdae', capacity: 10, type:
'Cabin' },
        { text: 'Rick Roll', id: 7, color: '#865fcf', capacity: 20, type:
'Conference' },
        { text: 'Rainbow', id: 8, color: '#1aaa55', capacity: 8, type:
'Cabin' },
        { text: 'Swarm', id: 9, color: '#df5286', capacity: 30, type:
'Conference' },

```

```

        { text: 'Photogenic', id: 10, color: '#710193', capacity: 25, type:
'Conference' }
    ];
    onRenderCell(args: RenderCellEventArgs):void {
        if (args.elementType === 'emptyCells' &&
args.element.classList.contains('e-resource-left-td')) {
            let target: HTMLElement = args.element.children[0] as
HTMLElement;
            target.innerHTML = '<div class="name">Rooms</div><div
class="type">Type</div><div class="capacity">Capacity</div>';
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Schedule with Multiple columns](images/schedule-multiple-columns.png)

Collapse/Expand child resources in timeline views

It is possible to expand and collapse the resources which have child resource in timeline views dynamically. By default, resources are in expanded state with their child resource. We can collapse and expand the child resources in UI by setting `expandedField` option as `false` whereas its default value is `true`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel, GroupModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { resourceData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,

```

```

    selector: "app-root",
    // specifies the template string for the Schedule component
    template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [currentView]='currentView' [eventSettings]="eventSettings"
[group]='group'>
    <e-resources>
        <e-resource field="RoomId" title="Room" name="Rooms"
            [dataSource]="roomDataSource"
            textField="RoomText" idField="Id" colorField="RoomColor"
expandedField="IsExpand">
        </e-resource>
        <e-resource field="OwnerId" title="Owner" name="Owners"
            [dataSource]="ownerDataSource"
[allowMultiple]="allowMultipleCategory"
            textField='OwnerText' idField='Id' groupIDField='OwnerGroupId'
colorField='OwnerColor'>
        </e-resource>
    </e-resources>
    </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 4);
    public currentView: string = 'TimelineWeek';
    public views: Array<string> = ['TimelineWeek', 'TimelineMonth',
'Agenda'];
    public eventSettings: EventSettingsModel = {
      dataSource: resourceData
    };
    public group: GroupModel = {
      resources: ['Rooms', 'Owners']
    };
    public roomDataSource: Object[] = [
      { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2', IsExpand: false },
      { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
    ];
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
      { OwnerText: 'Nancy', Id: 1, OwnerGroupId: 1, OwnerColor: '#ffaa00'
},
      { OwnerText: 'Steven', Id: 2, OwnerGroupId: 2, OwnerColor: '#f8a398'
},
      { OwnerText: 'Michael', Id: 3, OwnerGroupId: 1, OwnerColor: '#7499e1'
}
    ];
    allowMultipleCategory: any;
  }

```

MAIN.TS

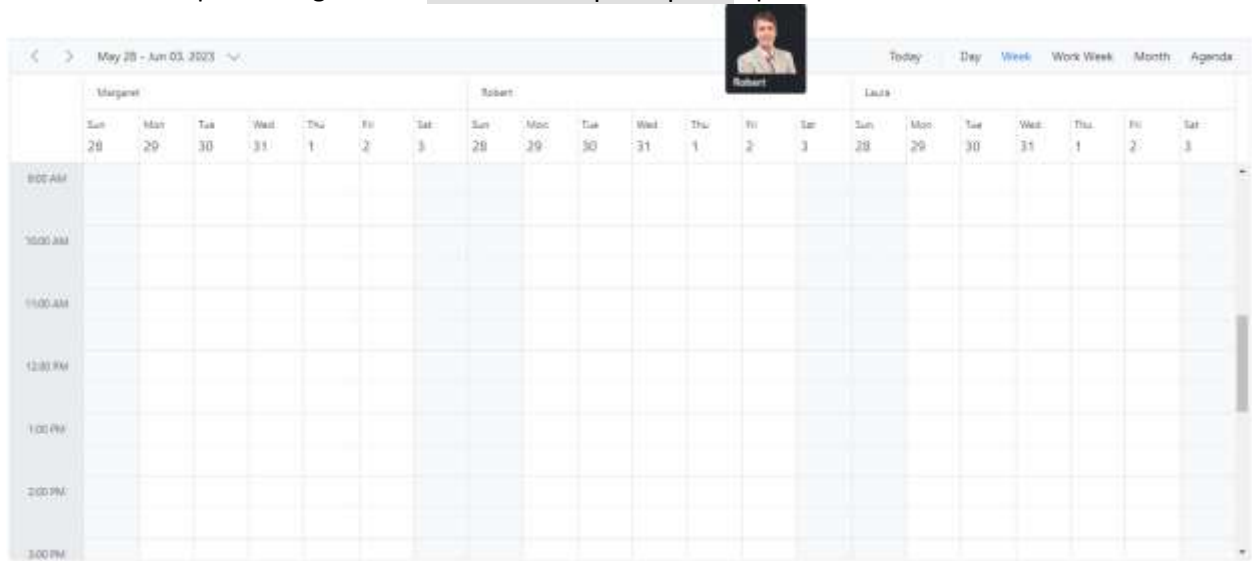
```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Displaying tooltip for resource headers

It is possible to display tooltips over the resource headers showing the resource information. By default, there won't be any tooltips displayed on the resource headers, and to enable it, you need to assign the customized template design to the `headerTooltipTemplate` option within the



group property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel, GroupModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { resourceData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService,
    TimelineMonthService ],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
    [views]="views"
  `
})
```

```

[eventSettings]="eventSettings" [group]='group'>
  <e-resources>
    <e-resource field="OwnerId" title="Owner" name="Owners"
      [dataSource]="roomDataSource" [allowMultiple]='allowMultipleRoom'
      textField='OwnerText' idField='Id' groupIdField='OwnerGroupId'
colorField='OwnerColor'>
    </e-resource>
  </e-resources>
  <ng-template #groupHeaderTooltipTemplate let-data>
    <div class='template-wrap'>
      <div class="res-text">Name:  {{data.resourceData.OwnerText}}
</div>
    </div>
  </ng-template>
</ejs-schedule>`
}))
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth'];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData
  };
  public group: GroupModel = {
    resources: ['Owners']
  };
  public allowMultipleRoom: Boolean = true;
  public roomDataSource: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerGroupId: 1, OwnerColor: '#ffaa00'
},
    { OwnerText: 'Steven', Id: 2, OwnerGroupId: 2, OwnerColor: '#f8a398'
},
    { OwnerText: 'Michael', Id: 3, OwnerGroupId: 1, OwnerColor: '#7499e1'
}
  ];
}

```

MAIN.TS

```

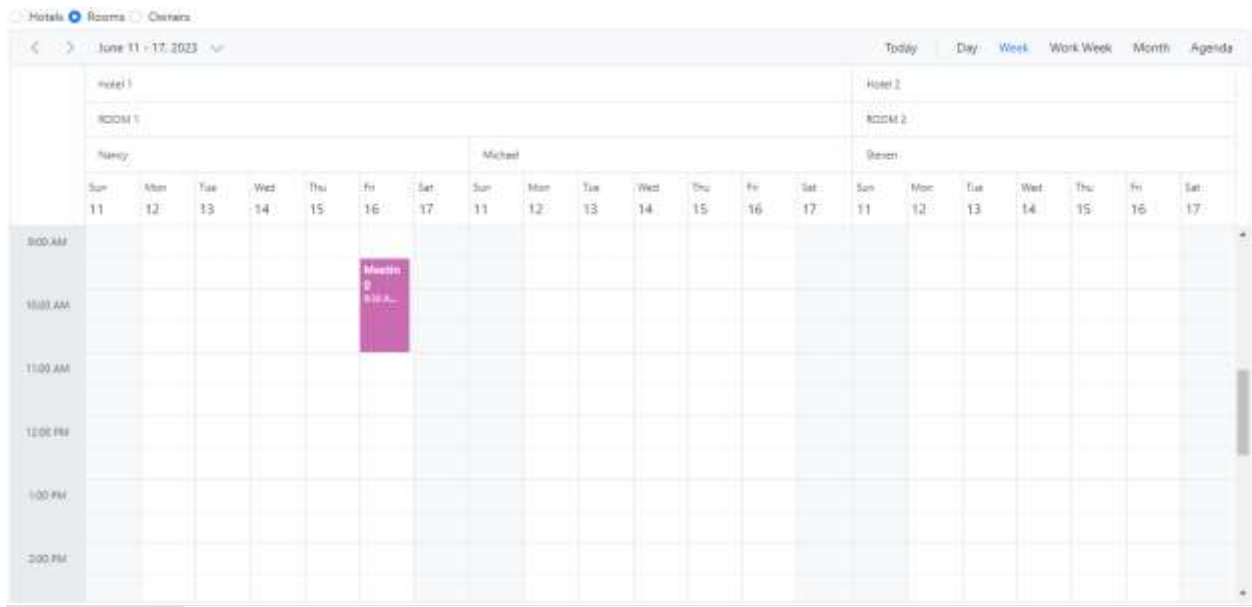
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![tooltip for resource headers](images/schedule-tooltip.png)

Choosing among resource colors for appointments

By default, the colors defined on the top level resources collection will be applied for the events. In case, if you want to apply specific resource color to events irrespective of its top-level parent resource color, it can be achieved by defining `resourceColorField` option within the



eventSettings property.

In the following example, the colors mentioned in the second level will get applied over the events.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import {
    DayService, ScheduleComponent, WorkWeekService, WeekService, MonthService,
    AgendaService, MonthAgendaService, TimelineViewsService,
    TimelineMonthService, EventSettingsModel, GroupModel
} from "@syncfusion/ej2-angular-schedule";
import { ChangeArgs } from "@syncfusion/ej2-buttons";
import { resourceData } from "../datasource";
@Component({
    imports: [
        ScheduleModule,
        RadioButtonModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        TimelineViewsService, TimelineMonthService],
    standalone: true,
    selector: "app-root",
    // specifies the template string for the Schedule component
    template: `<div style="padding: 10px;"><ejs-radiobutton label="Rooms"
name="default" value="Rooms" checked="true"
(change)="onChange($event)"></ejs-radiobutton>
```

```

    <ejs-radiobutton label="Owners" name="default" value="Owners"
    (change)="onChange($event)"></ejs-radiobutton>
  </div>
  <ejs-schedule #scheduleObj width="100%" height="550px"
  [selectedDate]="selectedDate" [views]="views"
    [currentView]="currentView" [eventSettings]="eventSettings"
  [group]="group">
    <e-resources>
      <e-resource field="RoomId" title="Room" name="Rooms"
  [dataSource]="roomDataSource"
        textField="RoomText" idField="Id" colorField="RoomColor">
      </e-resource>
      <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource"
  [allowMultiple]="allowMultipleCategory"
        textField="OwnerText" idField="Id" groupIDField="OwnerGroupId"
        colorField="OwnerColor">
      </e-resource>
    </e-resources>
  </ejs-schedule>
)
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ["Week", "Month", "TimelineWeek",
"TimelineMonth", "Agenda"];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData,
    resourceColorField: 'Rooms'
  };
  public group: GroupModel = {
    resources: ["Rooms", "Owners"]
  };
  public roomDataSource: Object[] = [
    { RoomText: "ROOM 1", Id: 1, RoomColor: "#cb6bb2" },
    { RoomText: "ROOM 2", Id: 2, RoomColor: "#56ca85" }
  ];
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { OwnerText: "Nancy", Id: 1, OwnerGroupId: 1, OwnerColor: "#ffaa00" },
    { OwnerText: "Steven", Id: 2, OwnerGroupId: 2, OwnerColor: "#f8a398" },
    { OwnerText: "Michael", Id: 3, OwnerGroupId: 1, OwnerColor: "#7499e1" }
  ];
  currentView: any;
  allowMultipleCategory: any;
  public onChange(args: ChangeArgs): void {
    this.scheduleObj!.eventSettings.resourceColorField = args.value;
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```



```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

![tooltip for resource headers](images/schedule-resource-color.png)

The value of the `resourceColorField` field should be mapped with the `name` value given within the `resources` property.

Dynamically add and remove resources

It is possible to add or remove the resources dynamically to and from the Scheduler respectively. In the following example, when the checkboxes are checked and unchecked, the respective resources gets added up or removed from the Scheduler layout. To add new resource dynamically, `addResource` method is used which accepts the arguments such as resource object, resource name (within which level, the resource object to be added) and index (position where the resource needs to be added).

To remove the resources dynamically, `removeResource` method is used which accepts the index (position from where the resource to be removed) and resource name (within which level, the resource object presents) as parameters.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { holidayData, birthdayData, companyData, personalData } from
'./datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
import { ScheduleComponent, EventSettingsModel, GroupModel,
TimelineViewsService, TimelineMonthService, ResizeService, DragAndDropService
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    CheckBoxModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService, ResizeService,
    DragAndDropService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `<div class="control-section">
<div class="col-lg-12 property-section">
  <table id="property" title="Show / Hide Resource">
    <tbody>
```

```

        <tr>
            <td>
                <ejs-checkbox cssClass="personal" label="My Calender"
value="1" [checked]="true" [disabled]="true"
(change)="onChange($event)"></ejs-checkbox>
            </td>
            <td>
                <ejs-checkbox cssClass="company" label="Company"
value="2" [checked]="false" (change)="onChange($event)"></ejs-checkbox>
            </td>
            <td>
                <ejs-checkbox cssClass="birthday" label="Birthday"
value="3" [checked]="false" (change)="onChange($event)"></ejs-checkbox>
            </td>
            <td>
                <ejs-checkbox cssClass="holiday" label="Holiday"
value="4" [checked]="false" (change)="onChange($event)"></ejs-checkbox>
            </td>
        </tr>
    </tbody>
</table>
</div>
<div>
    <ejs-schedule #scheduleObj cssClass='schedule-add-remove-resources'
width='100%' height='650px' [group]="group" [selectedDate]="selectedDate"
[eventSettings]="eventSettings">
        <e-resources>
            <e-resource field='CalendarId' title='Calendars'
[dataSource]='resourceDataSource' [allowMultiple]='allowMultiple'
name='Calendars'
                textField='CalendarText' idField='CalendarId'
colorField='CalendarColor'>
            </e-resource>
        </e-resources>
        <e-views>
            <e-view option="Month"></e-view>
            <e-view option="TimelineWeek"></e-view>
            <e-view option="TimelineMonth"></e-view>
        </e-views>
    </ejs-schedule>
</div>
</div>`
    })
    export class AppComponent {
        public calendarCollections: Object[] = [
            { CalendarText: 'My Calendar', CalendarId: 1, CalendarColor:
'#c43081' },
            { CalendarText: 'Company', CalendarId: 2, CalendarColor: '#ff7f50' },
            { CalendarText: 'Birthday', CalendarId: 3, CalendarColor: '#AF27CD' },
            { CalendarText: 'Holiday', CalendarId: 4, CalendarColor: '#808000' }
        ];
        public group: GroupModel = { resources: ['Calendars'] };
        public resourceDataSource: Object[] = [this.calendarCollections[0]];
        public allowMultiple: Boolean = true;
        public selectedDate: Date = new Date(2018, 3, 1);
    }

```

```

    public eventSettings: EventSettingsModel = { dataSource:
this.generateCalendarData() };
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    generateCalendarData(): Object[] {
        return [...personalData, ...companyData, ...birthdayData,
...holidayData];
    }
    public onChange(args: ChangeEventArgs): void {
        const value: number = parseInt((args.event?.currentTarget as
Element)?.querySelector('input')?.getAttribute('value') ?? '0', 10);
        const resourceData: Record<string, any>[] =
            this.calendarCollections.filter((calendar: Record<string, any>) =>
calendar['CalendarId'] === value);
        if (args.checked) {
            this.scheduleObj?.addResource(resourceData[0], 'Calendars', value
);
        } else {
            this.scheduleObj?.removeResource(value, 'Calendars');
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

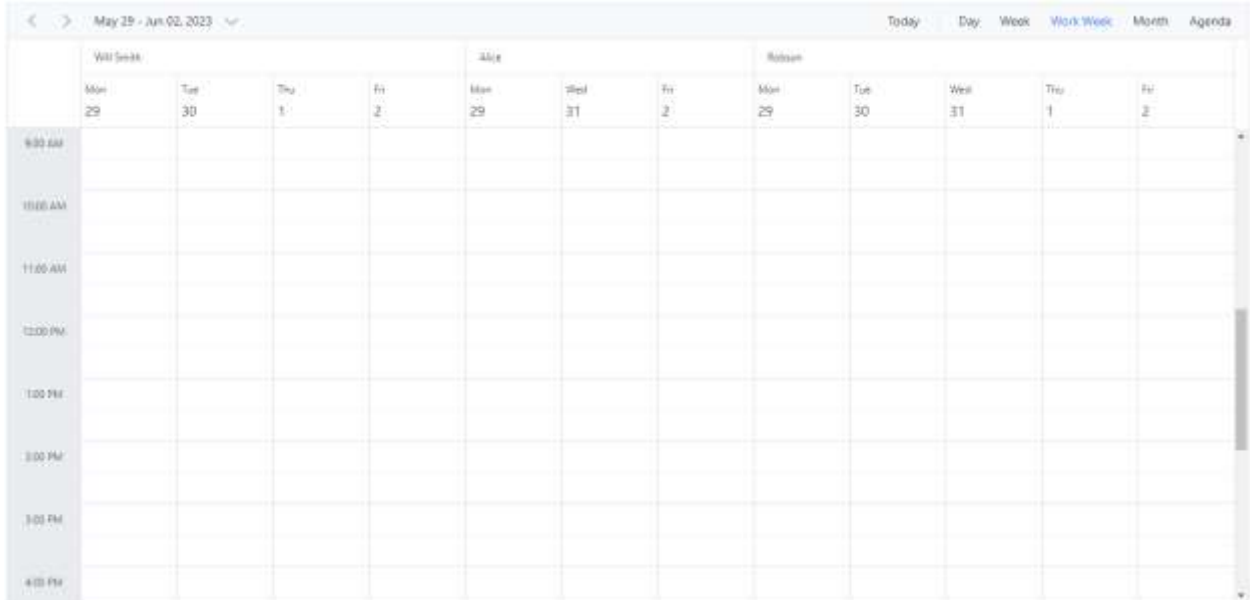
```

Setting different working days and hours for resources

Each resource in the Scheduler can have different working hours as well as different working days set to it. There are default options available within the `resources` collection, to customize the default working hours and days of the Scheduler.

Set different work days

Different working days can be set for the resources of Scheduler using the



workDaysField property which maps the working days field from the resource dataSource. This field accepts the collection of day indexes (from 0 to 6) of a week. By default, it is set to [1, 2, 3, 4, 5] and in the following example, each resource has been set with different values and therefore each of them will render only those working days. This option is applicable only on the calendar views and is not applicable on timeline views.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, EventSettingsModel, GroupModel } from
'@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { doctorData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    RadioButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
```

```

<ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
  [currentView]='currentView' [eventSettings]="eventSettings"
[group]='group'>
  <e-resources>
    <e-resource field="DoctorId" title="Conference" name="Conferences"
      [dataSource]="conferenceDataSource"
[allowMultiple]="allowMultipleCategory"
      textField='text' idField='id' colorField='color'
workDaysField='workDays'>
    </e-resource>
  </e-resources>
</ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 3, 1);
  public views: Array<string> = ['Week', 'WorkWeek', 'Month'];
  public currentView: string = 'WorkWeek';
  public eventSettings: EventSettingsModel = {
    dataSource: doctorData
  };
  public group: GroupModel = {
    resources: ['Conferences']
  };
  public allowMultipleCategory: Boolean = true;
  public conferenceDataSource: Object[] = [
    { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4, 5]
  },
    { text: 'Alice', id: 2, color: 'rgb(53, 124, 210)', workDays: [1, 3,
5] },
    { text: 'Robson', id: 3, color: '#7fa900' , workDays: [2,6]}
  ];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

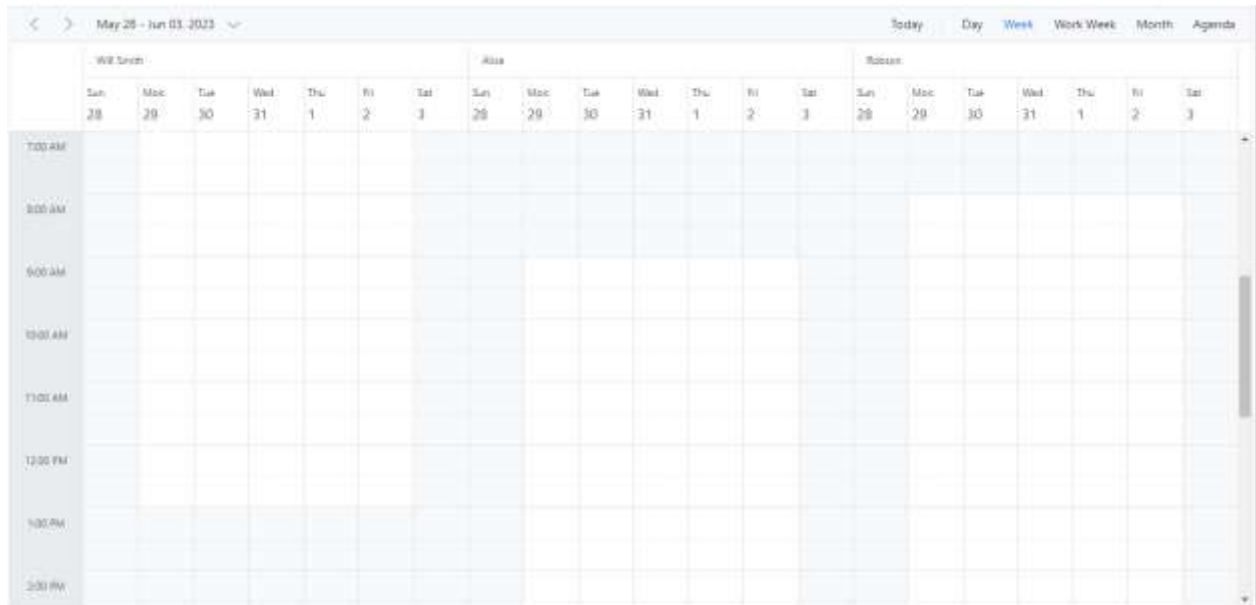
```

![Resources with Different Workdays](images/schedule-resource-workdays.png)

Set different work hours

Working hours indicates the work hour duration of a day, which is highlighted visually with active color over the work cells. Each resource on the Scheduler can be defined with its own set of working hours as depicted in the following example.

- **startHourField** - Denotes the start time of the working/business hour in a day.



• **endHourField** - Denotes the end time limit of the working/business hour in a day.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel, GroupModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { doctorData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    RadioButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
    [views]="views"
    [eventSettings]="eventSettings" [group]='group'>
    <e-resources>
      <e-resource field="DoctorId" title="Conference" name="Conferences"

```

```

        [dataSource]="conferenceDataSource"
    [allowMultiple]="allowMultipleCategory"
        textField='text' idField='id' colorField='color'
    startHourField='startHour' endHourField='endHour'>
    </e-resource>
</e-resources>
</ejs-schedule>`
    ))
    export class AppComponent {
        public selectedDate: Date = new Date(2018, 3, 1);
        public views: Array<string> = ['Week', 'Month', 'TimelineWeek',
'TimelineMonth'];
        public eventSettings: EventSettingsModel = {
            dataSource: doctorData
        };
        public group: GroupModel = {
            resources: ['Conferences']
        };
        public allowMultipleCategory: Boolean = true;
        public conferenceDataSource: Object[] = [
            { text: 'Will Smith', id: 1, color: '#ea7a57', startHour: '08:00',
endHour: '15:00' },
            { text: 'Alice', id: 2, color: '#357cd2', startHour: '10:00',
endHour: '18:00' },
            { text: 'Robson', id: 3, color: '#7fa900', startHour: '06:00',
endHour: '13:00' }
        ];
    }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Resources with Different Workhours](images/schedule-different-workhour.png)

In this example, a resource named **Will Smith** is depicted with working hours ranging from 8.00 AM to 3.00 PM and is visually illustrated with active colors, whereas the other two resources have different working hours set.

Hide non-working days when grouped by date

In Scheduler, you can set custom work days for each resource and group the Scheduler by date to display these work days. By default, the Scheduler will show all days when it is grouped by date, even if they are not included in the custom work days for the resources. However, you can use the [hideNonWorkingDays](#) property to only display the custom work days in the Scheduler.

To use the [hideNonWorkingDays](#) property, you need to include it in the configuration options for your Scheduler component. Set the value of [hideNonWorkingDays](#) to **true** to enable this feature.

Example: To display the Scheduler with resources grouped by date for custom working days,

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, EventSettingsModel, GroupModel } from
'@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [views]="views"
[group]='group'>
      <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
          [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
          textField='text' idField='id' colorField='color'
[workDaysField]='workDays'>
          </e-resource>
        </e-resources>
      </ejs-schedule>`
})
export class AppComponent {
  public views: Array<string> = ['Week', 'Month', 'Agenda'];
  public group: GroupModel = {
    byDate: true,
    hideNonWorkingDays: true,
    resources: ['Owners']
  };
  public allowMultipleOwner: Boolean = true;
  public ownerDataSource: Object[] = [
    { text: 'Alice', id: 1, color: '#1aaa55', workDays: [1, 2, 3, 4] },
    { text: 'Smith', id: 2, color: '#7fa900', workDays: [2, 3, 5] }
  ];
  workDays: any;
}

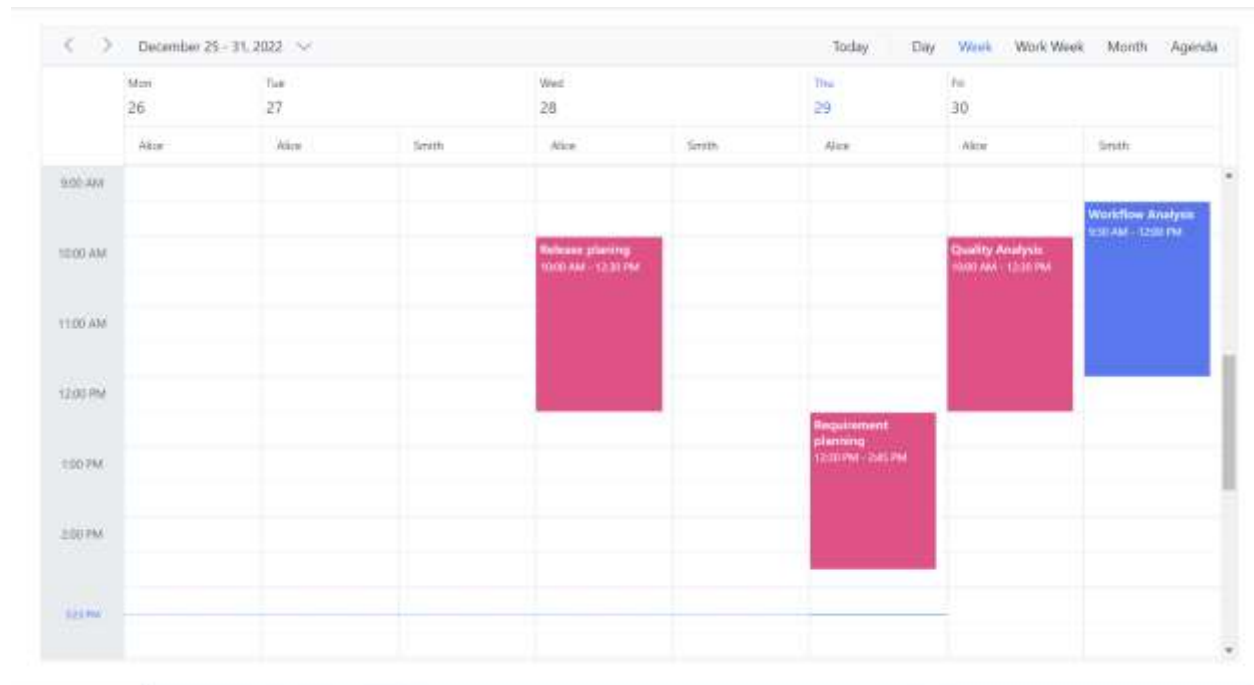
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

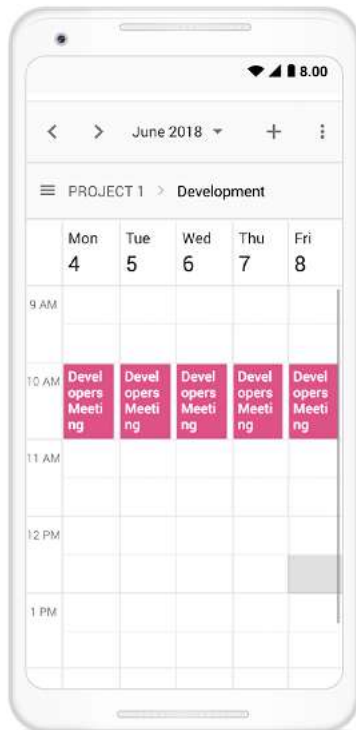



The [hideNonWorkingDays](#) property only applies when the Scheduler is grouped [byDate](#).

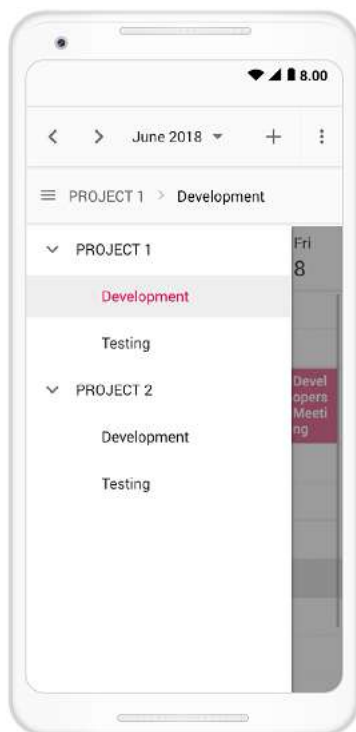
Compact view in mobile

Although the Scheduler views are designed keeping in mind the responsiveness of the control in mobile devices, however when using Scheduler with multiple resources - it is difficult to view all the resources and its relevant events at once on the mobile. Therefore, we have introduced a new compact mode specially for displaying multiple resources of Scheduler on mobile devices. By default, this mode is enabled while using Scheduler with multiple resources on mobile devices. If in case, you need to disable this compact mode, set `false` to the `enableCompactView` option within the `group` property. Disabling this option will display the exact desktop mode of Scheduler view on mobile devices.

With this compact view enabled on mobile, you can view only single resource at a time and to switch to other resources, there is a treeview at the left listing out all other available resources - clicking on which will display that particular resource and its related appointments.



Clicking on the menu icon before the resource text will show the resources available in the Scheduler as following.



[Adaptive UI in desktop](#)

By default, the Scheduler layout adapts automatically in the desktop and mobile devices with appropriate UI changes. In case, if the user wants to display the Adaptive scheduler in desktop mode

with adaptive enhancements, then the property `enableAdaptiveUI` can be set to true. Enabling this option will display the exact mobile mode of Scheduler view on desktop devices.

Some of the default changes made for compact Scheduler to render in desktop devices are as follows,

- View options displayed in the Navigation drawer.
- Plus icon is added to the header for new event creation.
- Today icon is added to the header instead of the Today button.
- With Multiple resources – only one resource has been shown to enhance the view experience of resource events details clearly. To switch to other resources, there is a TreeView on the left that lists all other available resources, clicking on which will display that particular resource and its related events.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, DragAndDropService, View,
ScheduleComponent, EventSettingsModel, GroupModel, } from '@syncfusion/ej2-
angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { extend } from '@syncfusion/ej2-base';
import { resourceData, resourceConferenceData } from '../datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  standalone: true,
  selector: "app-root",
  providers: [DayService, WeekService, MonthService, DragAndDropService,
WorkWeekService, AgendaService,
MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule #scheduleObj id="schedule" width='100%' height='650px'
[group]="group" [(currentView)]="currentView"
[selectedDate]="selectedDate" [enableAdaptiveUI]="true"
[eventSettings]="eventSettings">
      <e-views>
        <e-view option="Day"></e-view>
        <e-view option="Week"></e-view>
        <e-view option="Month"></e-view>
      </e-views>
      <e-resources>
        <e-resource field='ProjectId' title='Choose Project'
[dataSource]='projectDataSource'
[allowMultiple]='allowMultiple' name='Projects' textField='text'
idField='id' colorField='color'>
          </e-resource>
```

```

        <e-resource field='TaskId' title='Category'
[dataSource]='categoryDataSource' [allowMultiple]='allowMultiple'
        name='Categories' textField='text' idField='id'
groupIDField='groupId' colorField='color'>
        </e-resource>
    </e-resources>
</ejs-schedule>`
    })
    export class AppComponent {
        @ViewChild('scheduleObj')
        public scheduleObj?: ScheduleComponent;
        public selectedDate: Date = new Date(2018, 3, 4);
        public currentView: View = 'Month';
        public group: GroupModel = {
            resources: ['Projects', 'Categories']
        };
        public projectDataSource: Object[] = [
            { text: 'PROJECT 1', id: 1, color: '#cb6bb2' },
            { text: 'PROJECT 2', id: 2, color: '#56ca85' },
            { text: 'PROJECT 3', id: 3, color: '#df5286' }
        ];
        public categoryDataSource: Object[] = [
            { text: 'Nancy', id: 1, groupId: 1, color: '#df5286' },
            { text: 'Steven', id: 2, groupId: 1, color: '#7fa900' },
            { text: 'Robert', id: 3, groupId: 2, color: '#ea7a57' },
            { text: 'Smith', id: 4, groupId: 2, color: '#5978ee' },
            { text: 'Michael', id: 5, groupId: 3, color: '#df5286' },
            { text: 'Root', id: 6, groupId: 3, color: '#00bdae' }
        ];
        public allowMultiple: Boolean = true;
        public eventSettings: EventSettingsModel = {
            dataSource: <Object[]>extend([], resourceData.concat(
resourceConferenceData), undefined, true)
        };
    }
}

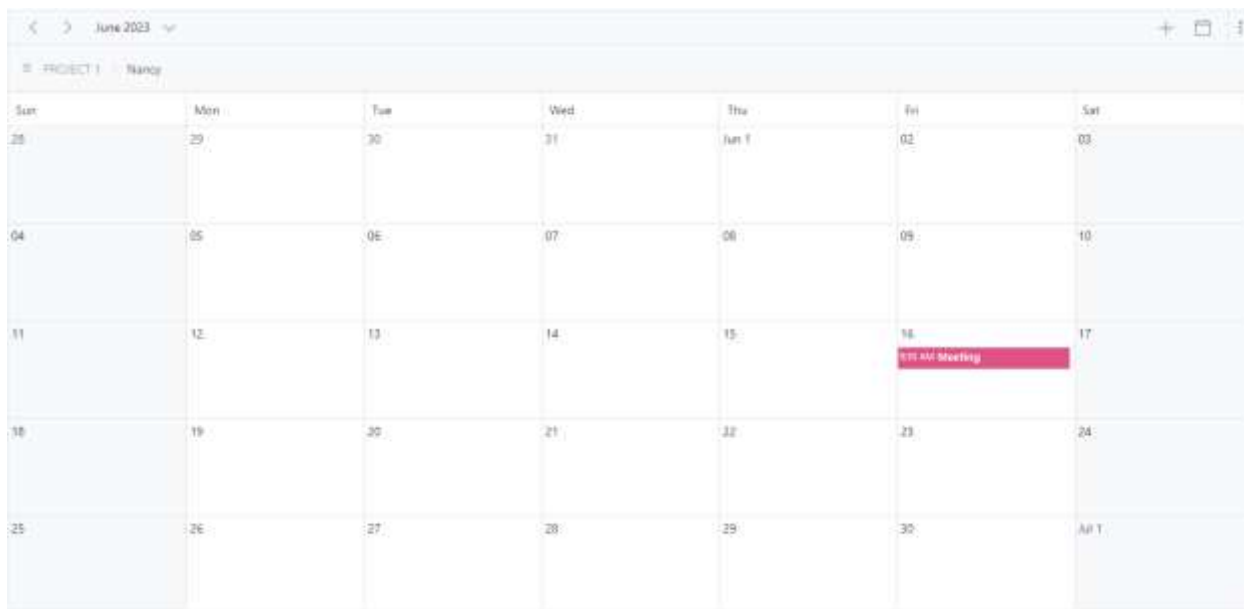
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Header rows in Angular Schedule component

The Timeline views can have additional header rows other than its default date and time header rows. It is possible to show individual header rows for displaying year, month and week separately using the **headerRows** property. This property is applicable only on the timeline views. The possible rows which can be added using **headerRows** property are as follows.

- Year
- Month
- Week
- Date
- Hour

The [Link to the Video](#) row is not applicable for Timeline month view.

Learn to add and customize additional header rows in the Timeline views of Angular Scheduler from this video:

The following example shows the Scheduler displaying all the available header rows on timeline views.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
```

```

import { EventSettingsModel, TimelineViewsService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
startHour='09:00' endHour='18:00'>
    <e-header-rows>
      <e-header-row option='Year'></e-header-row>
      <e-header-row option='Month'></e-header-row>
      <e-header-row option='Week'></e-header-row>
      <e-header-row option='Date'></e-header-row>
      <e-header-row option='Hour'></e-header-row>
    </e-header-rows>
    <e-views>
      <e-view option='TimelineWeek'></e-view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 11, 31);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

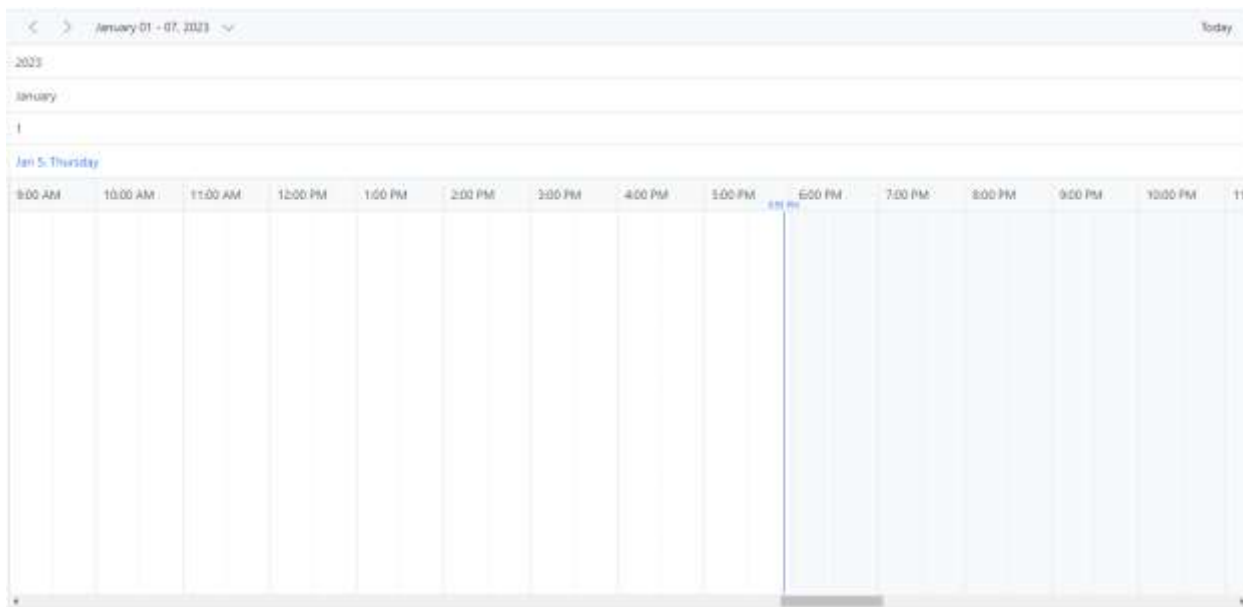
```

MAIN.TS

```

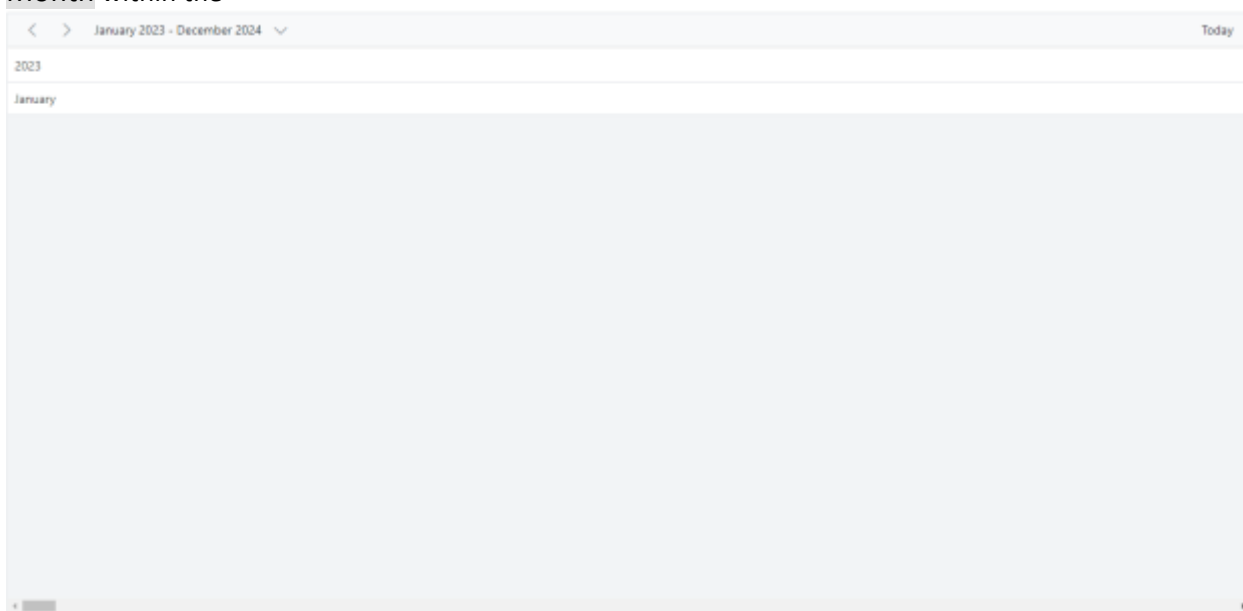
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Display year and month rows in timeline views

To display the timeline Scheduler simply with year and month names alone, define the option **Year** and **Month** within the



headerRows property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService } from '@syncfusion/ej2-
angular-schedule';
```

```

import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings">
    <e-header-rows>
      <e-header-row option='Year'></e-header-row>
      <e-header-row option='Month'></e-header-row>
    </e-header-rows>
    <e-views>
      <e-view option='TimelineMonth' [interval]="viewInterval"></e-
view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 11, 31);
  public viewInterval: Boolean | Number = 24;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

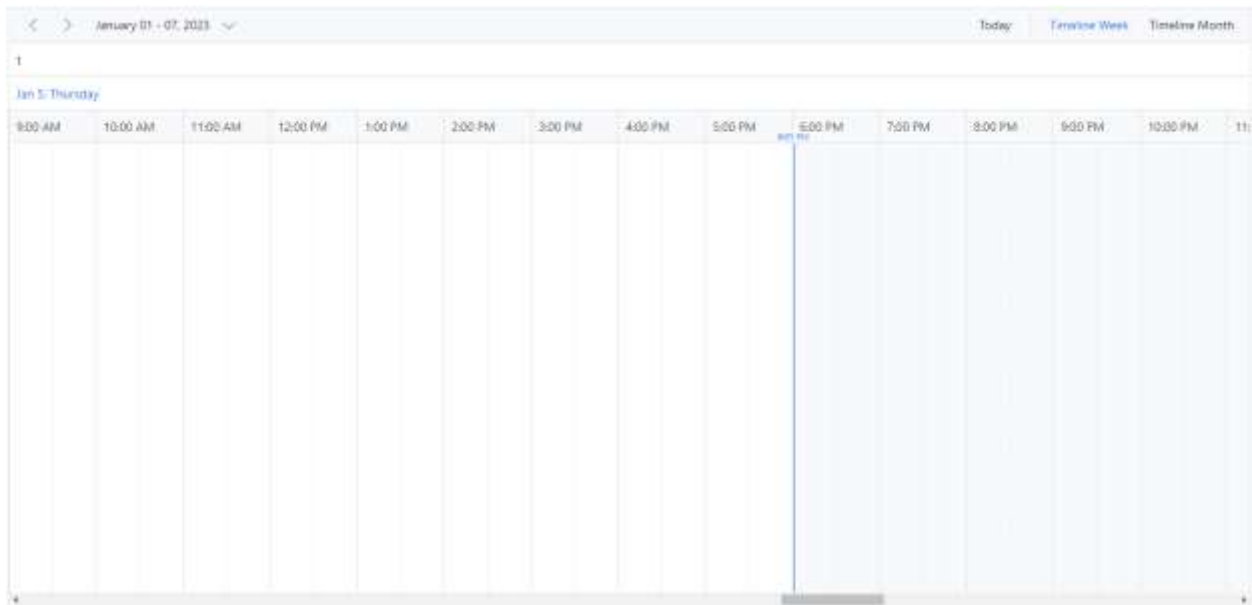
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Display Year and Month rows in timeline views](images/schedule-headerrow-month-year.png)

Display week numbers in timeline views

The week number can be displayed in a separate header row of the timeline Scheduler by setting **Week** option within



headerRows property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService, TimelineMonthService }
from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings">
    <e-header-rows>
      <e-header-row option='Week'></e-header-row>
      <e-header-row option='Date'></e-header-row>
      <e-header-row option='Hour'></e-header-row>
    </e-header-rows>`
```

```

    <e-views>
      <e-view option='TimelineMonth' [interval]="monthInterval"></e-
view>
      <e-view option='TimelineWeek' [interval]="weekInterval"></e-view>
      <e-view option='TimelineDay' [interval]="dayInterval"></e-view>
    </e-views>
  </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public monthInterval: Boolean | Number = 24;
    public weekInterval: Boolean | Number = 3;
    public dayInterval: Boolean | Number = 4;
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  }

```

MAIN.TS

```

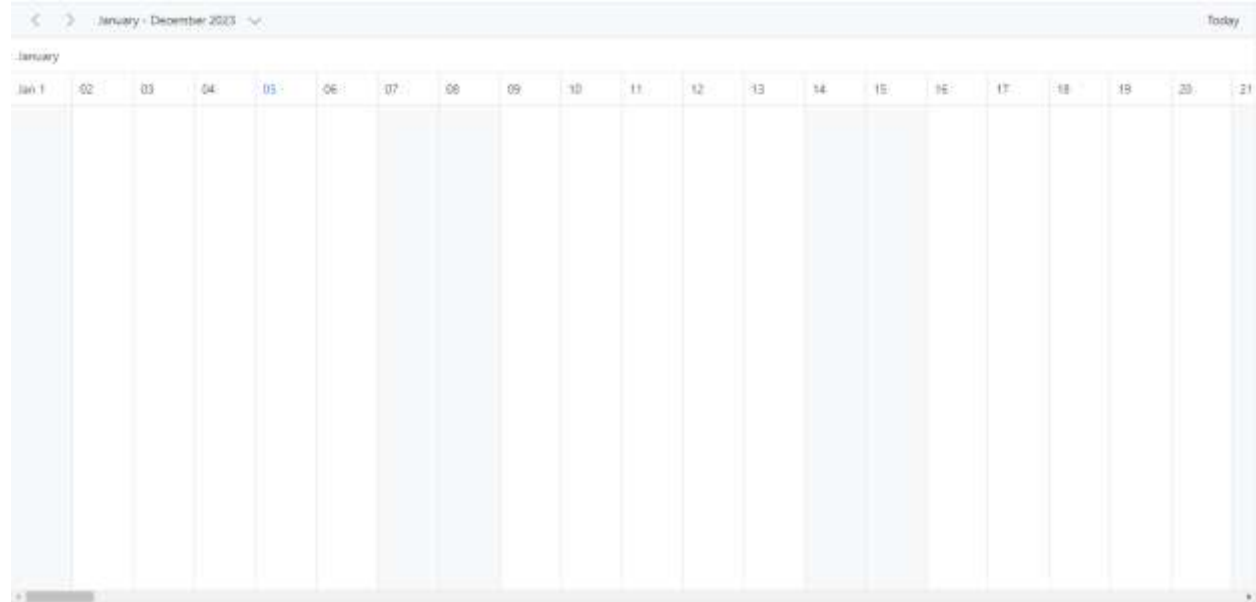
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

! [Display Week numbers in timeline views] (images/schedule-headerrow-weeknumber.png)

Timeline view displaying dates of a complete year

It is possible to display a complete year in a timeline view by setting `interval` value as 12 and defining



TimelineMonth view option within the `views` property of Scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';

```

```

import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings">
    <e-header-rows>
      <e-header-row option='Month'></e-header-row>
      <e-header-row option='Date'></e-header-row>
    </e-header-rows>
    <e-views>
      <e-view option='TimelineMonth' [interval]="viewInterval"></e-
view>
    </e-views>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 0, 1);
  public viewInterval: Boolean | Number = 12;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

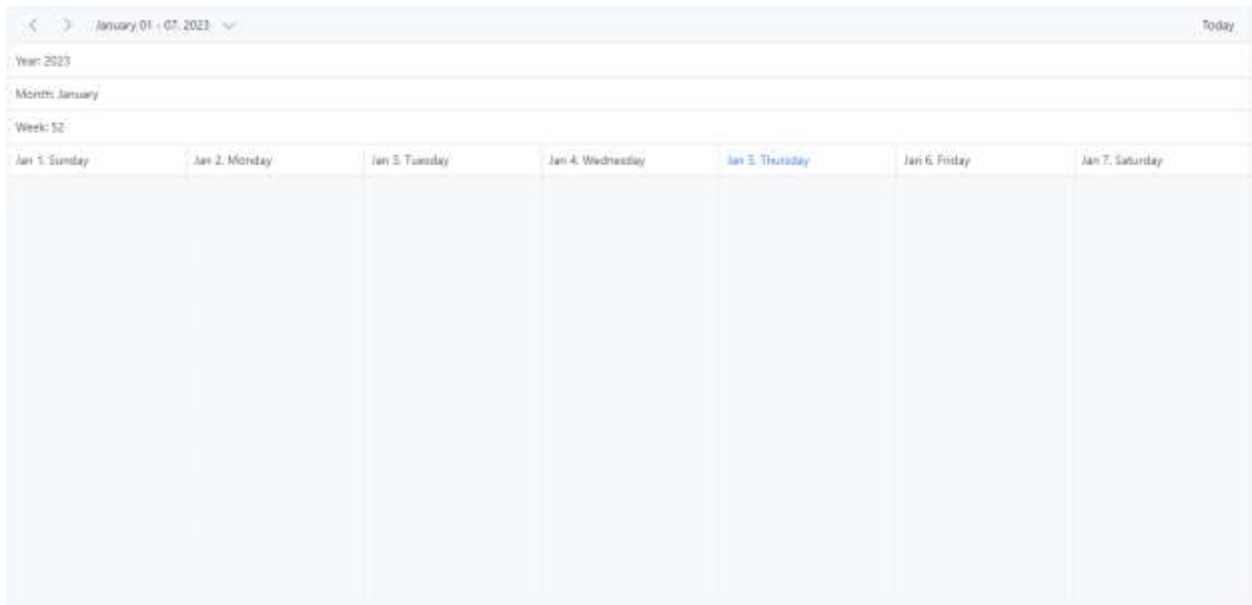
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Display dates of a complete year in timeline views](images/schedule-headerrow-dates.png)

Customizing the header rows using template

You can customize the text of the header rows and display any images or formatted text on each individual header rows using the built-in `template` option available within the



headerRows property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
import { EventSettingsModel, TimelineMonthService, getWeekNumber,
getWeekLastDate, CellTemplateArgs } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings">
    <e-header-rows>
      <e-header-row option='Year'>
        <ng-template #template let-data>
          <span [innerHTML]="getYearDetails(data)"></span>
        </ng-template>
      </e-header-row>
    </e-header-rows>
  `
})
export class AppComponent {}
```

```

        </ng-template>
      </e-header-row>
      <e-header-row option='Month'>
        <ng-template #template let-data>
          <span [innerHTML]="getMonthDetails(data)"></span>
        </ng-template>
      </e-header-row>
      <e-header-row option='Week'>
        <ng-template #template let-data>
          <span [innerHTML]="getWeekDetails(data)"></span>
        </ng-template>
      </e-header-row>
      <e-header-row option='Date'></e-header-row>
    </e-header-rows>
    <e-views>
      <e-view option='TimelineMonth'></e-view>
    </e-views>
  </ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 0, 1);
    public instance: Internationalization = new Internationalization();
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public getYearDetails(value: CellTemplateArgs): string {
      return 'Year: ' + this.instance.formatDate((value as
CellTemplateArgs).date, { skeleton: 'y' });
    }
    public getMonthDetails(value: CellTemplateArgs): string {
      return 'Month: ' + this.instance.formatDate((value as
CellTemplateArgs).date, { skeleton: 'yMMM' });
    }
    public getWeekDetails(value: CellTemplateArgs): string {
      return 'Week: ' + getWeekNumber(getWeekLastDate((value as
CellTemplateArgs).date, 0));
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

![Display customize header rows using template](images/schedule-headerrow-custom-header.png)

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Row auto height in Angular Schedule component

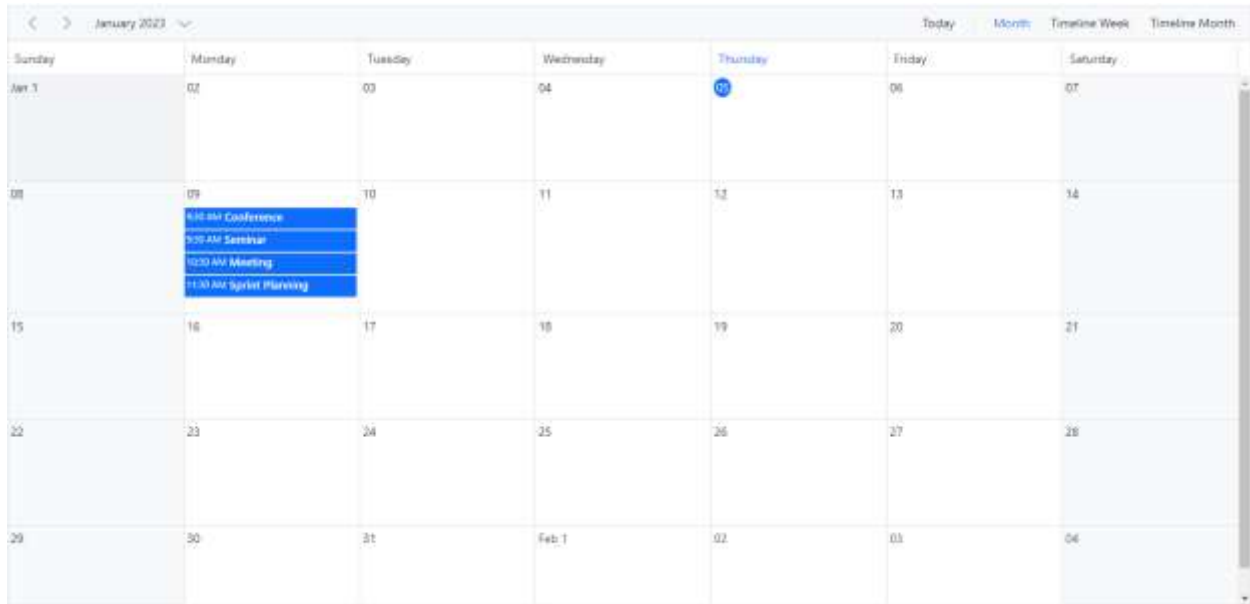
By default, the height of the Scheduler rows in Timeline views are static and therefore, when the same time range holds multiple overlapping appointments, a **+n more** text indicator will be displayed. With this feature enabled, you can now view all the overlapping appointments present in those specific time

range by auto-adjusting the row height based on the presence of the appointments count, instead of displaying the **+n more** text indicators.

To enable auto row height adjustments on Scheduler Timeline views and Month view, set **true** to the **rowAutoHeight** property whose default value is **false**.

This auto row height adjustment is applicable only on all the Timeline views as well as on the calendar Month view.

Now, let's see how it works on those applicable views with examples.



Calendar month view

By default, the rows of the calendar Month view can hold only the limited appointments count based on its row height, and the rest of the overlapping appointments are indicated with a **+n more** text indicator. The following example shows how the month view row auto-adjusts based on the number of appointments count, when this **RowAutoHeight** feature is enabled.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, View } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [rowAutoHeight]="rowAutoHeight"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[currentView]="currentView">
  <e-views><e-view option='Month' ></e-view></e-views></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleViews: View[] = ['Month'];
  public rowAutoHeight: boolean = true;
  public currentView: View = 'Month';
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Timeline views

When the feature **RowAutoHeight** is enabled in Timeline views, the row height gets auto-adjusted based on the number of overlapping events occupied on the same time range, which is demonstrated in the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService, TimelineMonthService, View
} from '@syncfusion/ej2-angular-schedule';
import { eventData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,

```

```

        AgendaService,
        MonthAgendaService,
        TimelineViewsService, TimelineMonthService,],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [rowAutoHeight]="rowAutoHeight"
[currentView]="currentView">
    <e-views> <e-view option='TimelineDay'></e-view>
    <e-view option='TimelineWeek'></e-view>
    <e-view option='TimelineWorkWeek'></e-view>
    <e-view option='TimelineMonth'></e-view>
    <e-view option='Agenda'></e-view>
</e-views></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public rowAutoHeight: boolean = true;
    public scheduleViews: View[] = ['TimelineWeek'];
    public currentView: View = 'TimelineWeek';
    public eventSettings: EventSettingsModel = { dataSource: eventData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Timeline views with multiple resources

The following example shows how the auto row adjustment feature works on timeline views with multiple resources.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, TimelineViewsService,
DragAndDropService, GroupModel, ResizeService, View, TimelineMonthService,
AgendaService } from '@syncfusion/ej2-angular-schedule';
import { roomData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, ResizeService, DragAndDropService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [group]="group" [currentView]="currentView"
[rowAutoHeight]="rowAutoHeight">
  <e-resources><e-resource field='RoomId' title='Room Type'
[dataSource]='resourceDataSource' [allowMultiple]='allowMultiple'
name='MeetingRoom' textField='text' idField='id' colorField='color'></e-
resource>
  <e-views> <e-view option='TimelineDay'></e-view>
    <e-view option='TimelineWeek'></e-view>
    <e-view option='TimelineWorkWeek'></e-view>
    <e-view option='TimelineMonth'></e-view>
    <e-view option='Agenda'></e-view></e-views></e-resources></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 7, 1);
  public rowAutoHeight: boolean = true;
  public scheduleViews: View[] = ['TimelineWeek'];
  public currentView: View = 'TimelineWeek';
  public group: GroupModel = {
    enableCompactView: false,
    resources: ['MeetingRoom']
  }
}
```

```

};
public allowMultiple: Boolean = true;
public resourceDataSource: Object[] = [
  { text: 'Room A', id: 1, color: '#98AFC7' },
  { text: 'Room B', id: 2, color: '#99c68e' },
  { text: 'Room C', id: 3, color: '#C2B280' },
  { text: 'Room D', id: 4, color: '#3090C7' },
  { text: 'Room E', id: 5, color: '#95b9' },
  { text: 'Room F', id: 6, color: '#95b9c7' },
  { text: 'Room G', id: 7, color: '#deb887' },
  { text: 'Room H', id: 8, color: '#3090C7' },
  { text: 'Room I', id: 9, color: '#98AFC7' },
  { text: 'Room J', id: 10, color: '#778899' }
];

public eventSettings: EventSettingsModel = {
  dataSource: roomData,
  fields: {
    id: 'Id',
    subject: { name: 'Subject', title: 'Summary' },
    location: { name: 'Location', title: 'Location' },
    description: { name: 'Description', title: 'Comments' },
    startTime: { name: 'StartTime', title: 'From' },
    endTime: { name: 'EndTime', title: 'To' }
  }
};
}

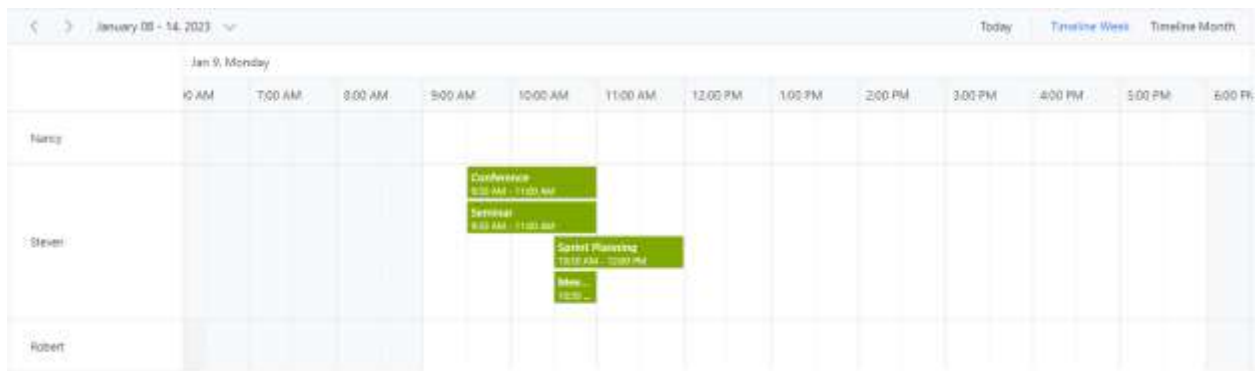
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Appointments occupying entire cell

By default, with the feature `rowAutoHeight`, there will be a space in the bottom of the cell when appointment is rendered. To avoid this space, we can set true to the property `ignoreWhitespace` with in `eventSettings` whereas its default property value is false. In the following code example, the whitespace below the appointments has been ignored.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel, GroupModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { resourceData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: "app-root",
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
    [currentView]='currentView' [rowAutoHeight]="rowAutoHeight"
[eventSettings]="eventSettings" [group]='group'>
      <e-resources>
        <e-resource field="RoomId" title="Room" name="Rooms"
          [dataSource]="roomDataSource"
          textField="RoomText" idField="Id" colorField="RoomColor">
        </e-resource>
        <e-resource field="OwnerId" title="Owner" name="Owners"
          [dataSource]="ownerDataSource"
[allowMultiple]="allowMultipleCategory"
          textField='OwnerText' idField='Id' groupIDField='OwnerGroupId'
colorField='OwnerColor'>
        </e-resource>
      </e-resources>
    </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2021, 7, 4);
  public currentView: string = 'TimelineWeek';
  public rowAutoHeight: boolean = true;
  public views: Array<string> = ['TimelineWeek', 'TimelineMonth'];
  public eventSettings: EventSettingsModel = {
    dataSource: resourceData,
    ignoreWhitespace: true
  };
  public group: GroupModel = {
    resources: ['Rooms', 'Owners']
  };
};

```

```

public roomDataSource: Object[] = [
  { RoomText: 'ROOM 1', Id: 1, RoomColor: '#cb6bb2' },
  { RoomText: 'ROOM 2', Id: 2, RoomColor: '#56ca85' }
];
public allowMultipleOwner: Boolean = true;
public ownerDataSource: Object[] = [
  { OwnerText: 'Nancy', Id: 1, OwnerGroupId: 1, OwnerColor: '#ffaa00' },
  { OwnerText: 'Steven', Id: 2, OwnerGroupId: 2, OwnerColor: '#f8a398' },
  { OwnerText: 'Michael', Id: 3, OwnerGroupId: 1, OwnerColor: '#7499e1' }
];
allowMultipleCategory: any;
}

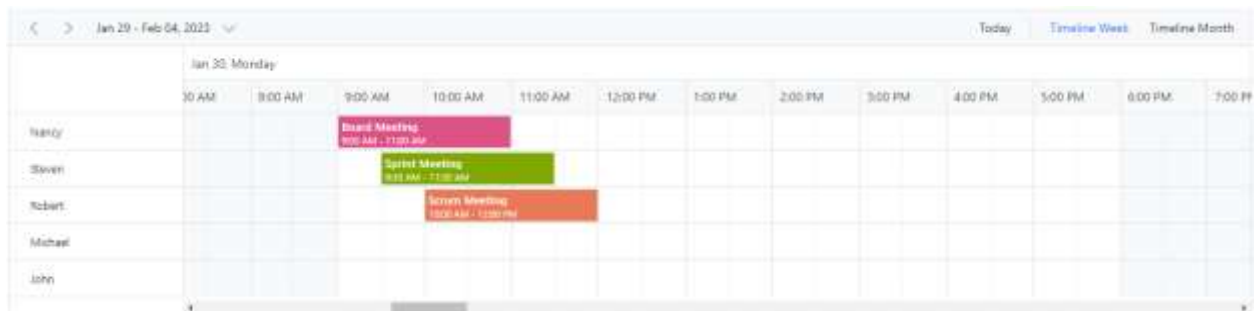
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Note: The property `ignoreWhitespace` will be applicable only when `rowAutoHeight` feature is enabled in the Scheduler.

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Header bar in Angular Schedule component

The header part of Scheduler can be customized easily with the built-in options available.

Show or Hide header bar

By default, the header bar holds the date and view navigation options, through which the user can switch between the dates and various views. This header bar can be hidden from the UI by setting `false` to the `showHeaderBar` property. Its default value is `true`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';

```

```

import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, EventSettingsModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showHeaderBar]='showHeaderBar'>
  </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  public showHeaderBar: Boolean = false;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Customizing header bar

Apart from the default date navigation and view options available on the header bar, you can add custom items into the Scheduler header bar by making use of the `actionBegin` event. Here, an employee image is added to the header bar, clicking on which will open the popup showing that person's short profile information.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { createElement, compile } from '@syncfusion/ej2-base';
import { ItemModel } from '@syncfusion/ej2-navigations';
import { Popup } from '@syncfusion/ej2-popups';
import { EventSettingsModel, ActionEventArgs, ToolbarActionArgs,
ScheduleComponent } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
```

```

    template: `<ejs-schedule id='schedule' width='100%' height='550px'
[selectedDate]='selectedDate' [views]='views' [eventSettings]='eventSettings'
[showHeaderBar]='showHeaderBar'
    [currentView]='currentView' (actionBegin)='onActionBegin($event)'
    (actionComplete)='onActionComplete($event)' #schedule></ejs-schedule>`
  })
  export class AppComponent {
    @ViewChild('schedule') public scheduleObj?: ScheduleComponent
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Month'];
    public currentView: string = 'Month';
    public eventSettings: EventSettingsModel = {
      dataSource: scheduleData
    };
    public userContentEle: HTMLElement = createElement('div', {
      className: 'e-profile-wrapper'
    });
    public profilePopup?: Popup;
    showHeaderBar: any;
    onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
      if (args.requestType === 'toolbarItemRendering') {
        let userIconItem: ItemModel = {
          align: 'Right', prefixIcon: 'user-icon', text: 'Nancy',
          cssClass: 'e-schedule-user-icon'
        };
        (args.items as ItemModel[]).push(userIconItem);
      }
    }
    onActionComplete(args: ActionEventArgs): void {
      let scheduleElement: HTMLElement = (this.scheduleObj as
ScheduleComponent).element;
      if (args.requestType === 'toolBarItemRendered') {
        let userIconEle: HTMLElement = scheduleElement.querySelector('.e-
schedule-user-icon') as HTMLElement;
        userIconEle.onclick = () => {
          (this.profilePopup as Popup).relateTo = userIconEle;
          (this.profilePopup as Popup).dataBind();
          if ((this.profilePopup as
Popup).element.classList.contains('e-popup-close')) {
            (this.profilePopup as Popup).show();
          } else {
            (this.profilePopup as Popup).hide();
          }
        };
      }
      let userContentEle: HTMLElement = createElement('div', {
        className: 'e-profile-wrapper'
      });
      (scheduleElement.parentElement as
HTMLElement).appendChild(userContentEle);
      let userIconEle: HTMLElement = scheduleElement.querySelector('.e-
schedule-user-icon') as HTMLElement;
      let getDOMString: (data: object) => NodeList = compile('<div
class="profile-container"><div class="profile-image">' +
        '</div><div class="content-wrap"><div class="name">Nancy</div>' +
        '<div class="destination">Product Manager</div><div
class="status">' +

```

```

        '<div class="status-icon"></div>Online</div></div></div>');
    let output: NodeList = getDOMString({});
    this.profilePopup = new Popup(userContentEle, {
        content: output[0] as HTMLElement,
        relateTo: userIconEle,
        position: { X: 'left', Y: 'bottom' },
        collision: { X: 'flip', Y: 'flip' },
        targetType: 'relative',
        viewPortElement: scheduleElement,
        width: 185,
        height: 80
    });
    this.profilePopup.hide();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to display the view options within the header bar popup

By default, the header bar holds the view navigation options, through which the user can switch between various views. You can move this view options to the header bar popup by setting `true` to the `enableAdaptiveUI` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component

```



```

    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
[enableAdaptiveUI]="enableAdaptiveUI"></ejs-schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: scheduleData,
    };
    public enableAdaptiveUI?: true;
  }

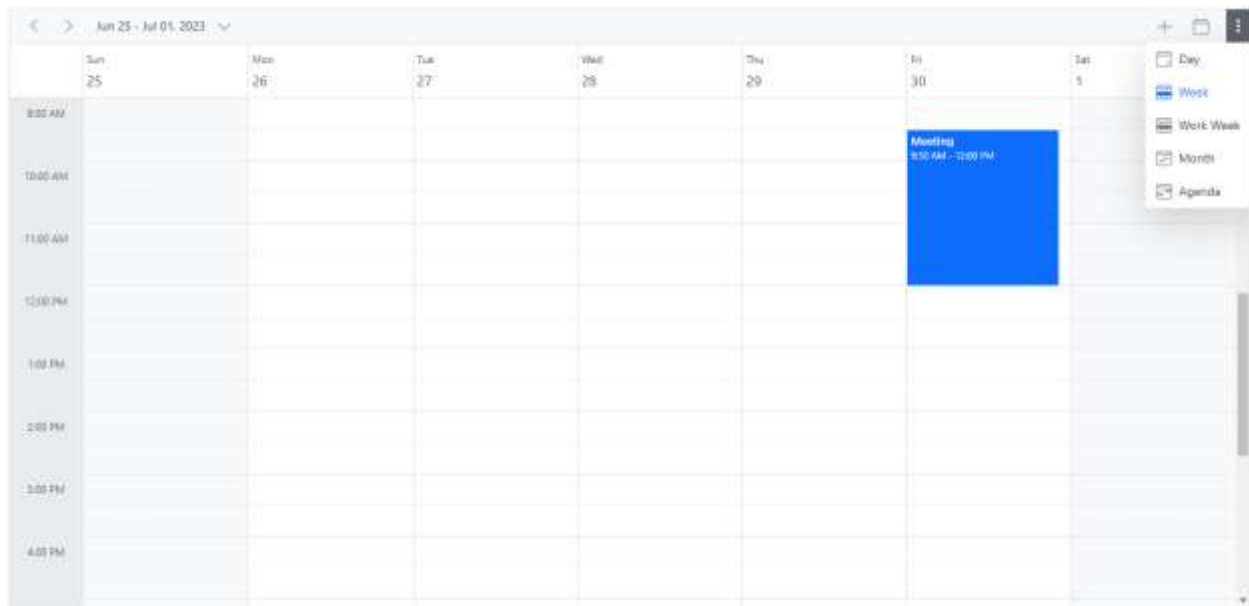
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Refer [here](#) to know more about adaptive UI in resources scheduler.

Date header customization

The Scheduler UI that displays the date text on all views are considered as the date header cells. You can customize the date header cells of Scheduler either using `dateHeaderTemplate` or `renderCell` event.

Using date header template

The `dateHeaderTemplate` option is used to customize the date header cells of day, week and work-week views.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';

```

```

import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService, EventSettingsModel } from '@syncfusion/ej2-angular-
schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService, TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule id='schedule' width='100%' height='550px'
[cssClass]='cssClass'
[selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'>
  <ng-template #dateHeaderTemplate let-data>
    <div class="date-text">{{getDateHeaderText(data.date)}}</div>
    <div [innerHTML]="getWeather(data.date)"></div>
  </ng-template>
</ejs-schedule>`,
  styles: [`.weather-text {
    padding: 5px;
    color: #e3165b;
    font-weight: 500;
  }`],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'Agenda',
'TimelineWorkWeek', 'TimelineMonth'];
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
  public cssClass: string = 'schedule-date-header-template';
  public instance: Internationalization = new Internationalization();
  getDateHeaderText: Function = (value: Date) => {
    return this.instance.formatDate(value, { skeleton: 'Ed' });
  };
  getWeather: Function = (value: Date) => {
    switch (value.getDay()) {
      case 0:
        return '<div class="weather-text">25°C</div>';
      case 1:

```

```

        return '<div class="weather-text">18°C</div>';
    case 2:
        return '<div class="weather-text">10°C</div>';
    case 3:
        return '<div class="weather-text">16°C</div>';
    case 4:
        return '<div class="weather-text">8°C</div>';
    case 5:
        return '<div class="weather-text">27°C</div>';
    case 6:
        return '<div class="weather-text">17°C</div>';
    default:
        return null;
    }
}
}

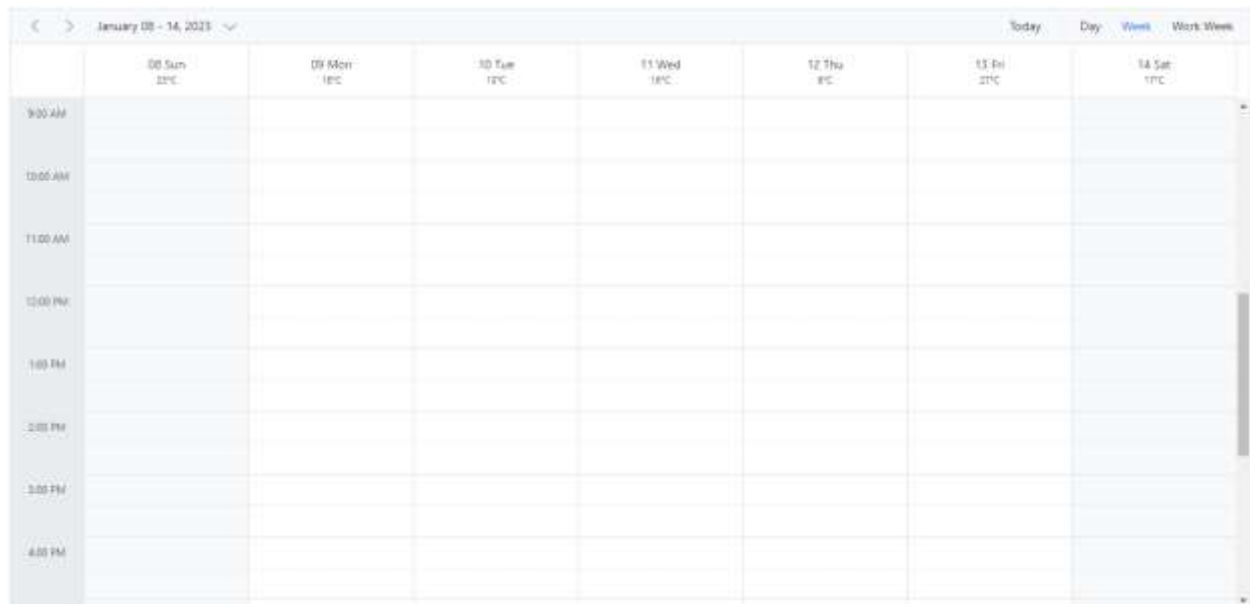
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Using renderCell event

In month view, the date header template is not applicable and therefore the same customization can be added beside the date text in month cells by making use of the `renderCell` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';

```

```

import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { EventSettingsModel, RenderCellEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
imports: [

    ScheduleModule,
    TimePickerModule
],
providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule id='schedule' width='100%' height='550px'
[cssClass]='cssClass' [selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'
(renderCell)='onRenderCell($event)'></ejs-schedule>`,
styles: [`.weather-text {
    float: right;
    margin: -20px 2px 0 0;
    color: #EA7A57;
}`],
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Month'];
    public eventSettings: EventSettingsModel = {
        dataSource: scheduleData
    };
    public cssClass: string = 'schedule-date-header-template';
    onRenderCell(args: RenderCellEventArgs): void {
        if (args.elementType === 'monthCells') {
            let ele: Element = document.createElement('div');
            ele.innerHTML = this.getWeather(args.date);
            (args.element).appendChild((ele as any).firstChild);
        }
    }
    getWeather: Function = (value: Date) => {
        switch (value.getDay()) {
            case 0:
                return '<div class="weather-text">25°C</div>';
            case 1:
                return '<div class="weather-text">18°C</div>';
            case 2:
                return '<div class="weather-text">10°C</div>';
            case 3:
                return '<div class="weather-text">16°C</div>';
            case 4:

```

```

        return '<div class="weather-text">8°C</div>';
      case 5:
        return '<div class="weather-text">27°C</div>';
      case 6:
        return '<div class="weather-text">17°C</div>';
      default:
        return null;
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing the date range text

The [dateRangeTemplate](#) option allows you to customize the text content of the date range displayed in the scheduler. By default, the date range text is determined by the scheduler view being used. However, you can use the [dateRangeTemplate](#) option to override the default text and specify your own custom text to be displayed.

The [dateRangeTemplate](#) property includes `startDate`, `endDate` and `currentView` options, you can customize the date range text using these available options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, TimelineViewsService,
TimelineMonthService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService,
    TimelineMonthService
  ],
  standalone: true,
  selector: 'app-root',

```

```
// specifies the template string for the Schedule component
template: `<ejs-schedule id='schedule' width='100%' height='650px'>
  <ng-template #dateRangeTemplate let-data>
    <div class="date-text">{{getDateRange(data.startDate)}}-
    {{getDateRange(data.endDate)}}</div>
  </ng-template>
  <e-views>
    <e-view option="Day"></e-view>
    <e-view option="Week"></e-view>
    <e-view option="WorkWeek"></e-view>
    <e-view option="Month"></e-view>
    <e-view option="Agenda"></e-view>
    <e-view option="TimelineMonth"></e-view>
  </e-views>
</ejs-schedule>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public instance: Internationalization = new Internationalization();
  public getDateRange(value: Date): string {
    return this.instance.formatDate(value, { skeleton: 'Ed' });
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Customizing header indent cells

It is possible to customize the header indent cells using the `headerIndentTemplate` option and change the look and appearance in both the vertical and timeline views. In vertical views, You can customize the header indent cells at the hierarchy level and you can customize the resource header left indent cell in timeline views using the template option.

Example: To customize the header left indent cell to display resources text, refer to the below code example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewEncapsulation } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
import { EventSettingsModel, DayService, WeekService, GroupModel,
TimelineViewsService, TimelineMonthService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { resourceData } from './datasource';
@Component({
  imports: [
```

```

        ScheduleModule,
        TimePickerModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService,
                TimelineViewsService, TimelineMonthService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule #scheduleObj width="100%" height="550px"
[selectedDate]="selectedDate" [views]="views"
[eventSettings]="eventSettings" [group]='group'>
    <e-resources>
        <e-resource field="OwnerId" title="Owner" name="Owners"
            [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
            textField='OwnerText' idField='Id' colorField='OwnerColor'>
        </e-resource>
    </e-resources>
    <ng-template #headerIndentTemplate let-data>
        <div class='e-resource-text'>
            <div class="text">Resources</div>
        </div>
    </ng-template>
</ejs-schedule>`,
    styleUrls: ['./index.css'],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 3, 1);
    public views: Array<string> = ['Day', 'Week', 'TimelineWeek',
'TimelineMonth'];
    public eventSettings: EventSettingsModel = {
        dataSource: resourceData
    };
    public group: GroupModel = {
        resources: ['Owners']
    };
    public allowMultipleOwner: Boolean = true;
    public ownerDataSource: Object[] = [
        { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
        { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
        { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }
    ];
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Timescale in Angular Schedule component

The time slots are usually the time cells that are displayed on the Day, Week and Work Week views of both the vertical views (to the left most position) and timeline views (at the top position). The **timeScale** property allows you to control and set the required time slot duration for the work cells displayed on Scheduler. It includes the following sub-options such as,

- **enable** - When set to **true**, allows the Scheduler to display the appointments accurately against the exact time duration. If set to **false**, all the appointments of a day will be displayed one below the other with no grid lines displayed. Its default value is **true**.
- **interval** – Defines the time duration on which the time axis to be displayed either in 1 hour or 30 minutes interval and so on. It accepts the values in minutes and defaults to 60.
- **slotCount** – Decides the number of slot count to be split for the specified time interval duration. It defaults to 2, thus displaying two slots to represent an hour(each slot depicting 30 minutes duration).

Note: The upper limit for rendering slots within a single day, utilizing the **interval** and **slotCount** properties of the **timeScale**, stands at 1000. This constraint aligns with the maximum **colspan** value permissible for the **table** element, also capped at 1000. This particular restriction is relevant exclusively to the **TimelineDay**, **TimelineWeek** and **TimelineWorkWeek** views.

Setting different time slot duration

The **interval** and **slotCount** properties can be used together on the Scheduler to set different time slot duration which is depicted in the following code example. Here, six time slots together represents an hour.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
```



```

MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[timeScale]="timeScale" > </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public timeScale: TimeScaleModel = { enable: true, interval: 60,
slotCount: 6 };
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

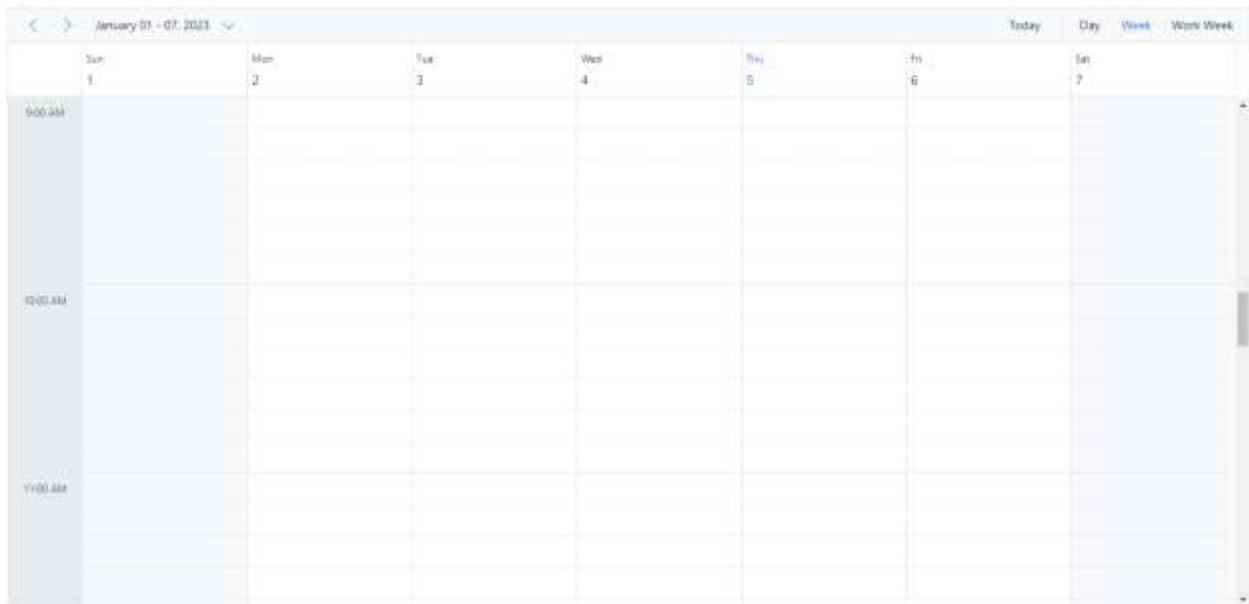
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Customizing time cells using template

The **timeScale** property also provides template option to allow customization of time slots which are as follows,

- **majorSlotTemplate** - The template option to be applied for major time slots. Here, the template accepts either the string or HTML element as template design and then the parsed design is displayed onto the time cells. The time details can be accessed within this template.
- **minorSlotTemplate** - The template option to be applied for minor time slots. Here, the template accepts either the string or HTML element as template design and then the parsed design is displayed onto the time cells. The time details can be accessed within this template.

APP.COMPONENT.HTML

```

<ejs-schedule width='100%' height='550px' [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [timeScale]="timeScale">
  <ng-template #timeScaleMajorSlotTemplate let-data>
    <div class="majorTime">{{getMajorTime(data.date)}}</div>
  </ng-template>
  <ng-template #timeScaleMinorSlotTemplate let-data>
    <div class="minorTime">{{getMinorTime(data.date)}}</div>
  </ng-template>
</ejs-schedule>

```

APP.COMPONENT.TS

```

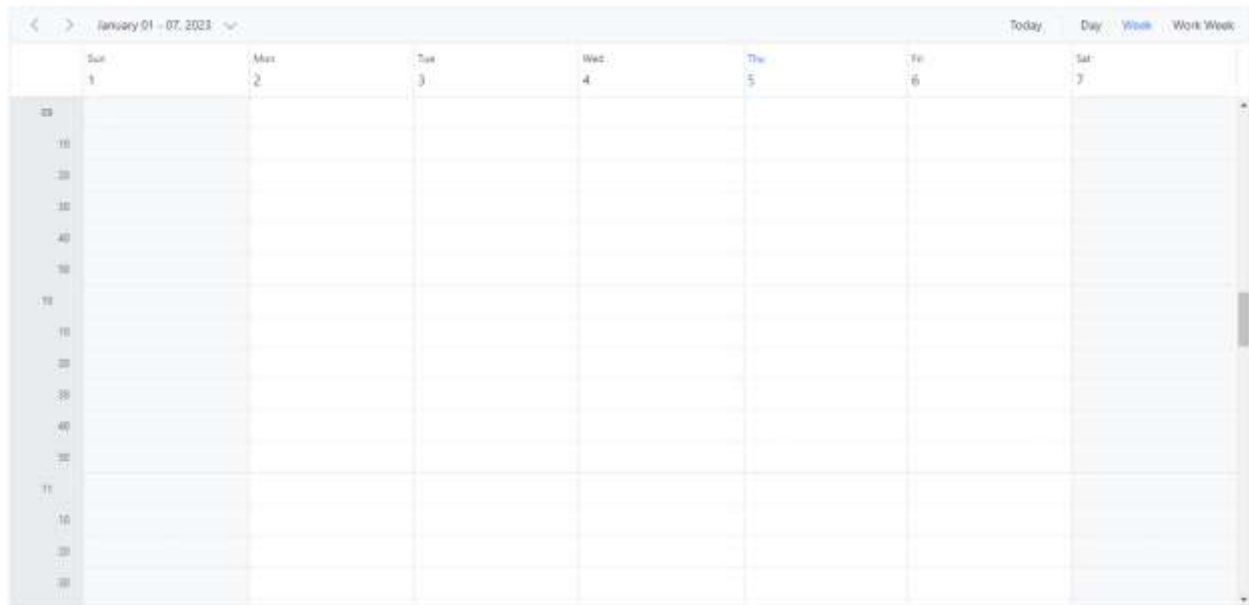
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel, DayService, WeekService,
WorkWeekService, MonthService, AgendaService, MonthAgendaService } from
'@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
  // specifies the template string for the Schedule component
  templateUrl: './app.component.html'
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public timeScale: TimeScaleModel = {
    enable: true,
    interval: 60,
    slotCount: 6,
    majorSlotTemplate: '#majorSlotTemplate',
    minorSlotTemplate: '#minorSlotTemplate'
  };
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public instance: Internationalization = new Internationalization();
  getMajorTime(date: Date): string {
    return this.instance.formatDate(date, { skeleton: 'hm' });
  }
  getMinorTime(date: Date): string {
    return this.instance.formatDate(date, { skeleton: 'ms'
  }).replace(':00', '');
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

**Hide the timescale**

The grid lines which indicates the exact time duration can be enabled or disabled on the Scheduler, by setting **true** or **false** to the **enable** option within the **timeScale** property. It's default value is **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
```

```

        MonthAgendaService],
standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[timeScale]="timeScale" > </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public timeScale: TimeScaleModel = { enable: false };
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

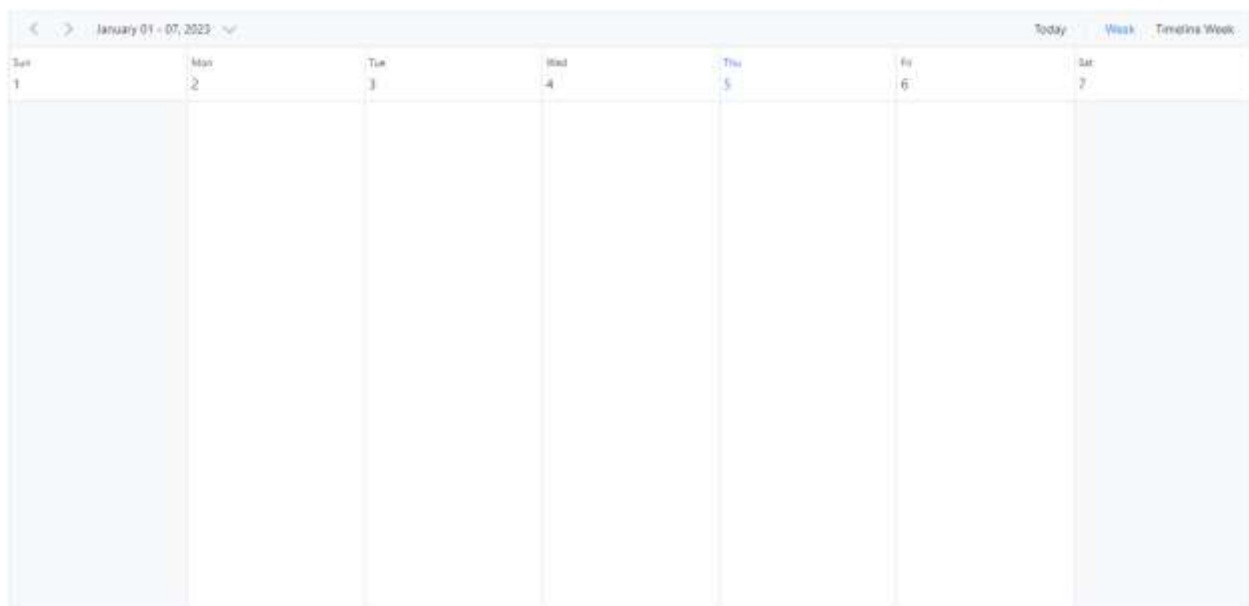
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Highlighting current date and time

By default, Scheduler indicates current date with a highlighted date header on all views, as well as marks accurately the system's current time on specific views such as Day, Week, Work Week, Timeline Day, Timeline Week and Timeline Work Week views. To stop highlighting the current time indicator on Scheduler views, set `false` to the `showTimeIndicator` property which defaults to `true`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'

```

```

import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-
schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[eventSettings]="eventSettings" [showTimeIndicator]="showTimeIndicator" >
</ejs-schedule>`
})
export class AppComponent {
  public showTimeIndicator: boolean = true;
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

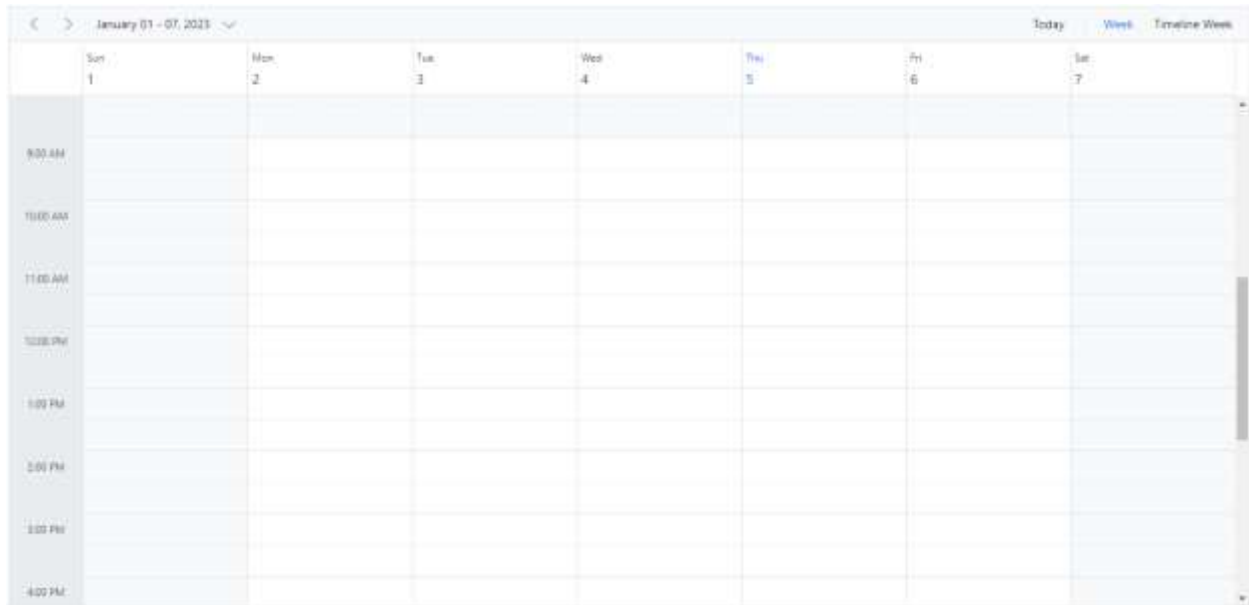
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Working days in Angular Schedule component

The Scheduler can be customized on various aspects as well as it inherits almost all the calendar-specific features such as options,

- To set custom time range display on Scheduler
- To set different working hours
- To set different working days
- To set different first day of week
- To show/hide weekend days
- To show the week number

Set working days

By default, Scheduler considers the week days from Monday to Friday as **Working days** and therefore defaults to [1,2,3,4,5] - where 1 represents Monday, 2 represents Tuesday and so on. The days which are not defined in this working days collection are considered as non-working days. Therefore, when the weekend days are set to hide from Scheduler, all those non-working days too get hidden from the layout.

The Work week and Timeline Work week views display exactly the defined working days on Scheduler layout, whereas other views display all the days and simply differentiate the non-working days on UI with inactive cell color.

The working or business hours depiction on Scheduler are usually valid only on these specified working days.

The following example code depicts how to set the Scheduler to display Monday, Wednesday and Friday as working days of a week.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineViewsService, View } from
'@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' currentView='WorkWeek'
[views]="scheduleViews" [workDays]='workWeekDays'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public workWeekDays: number[] = [1, 3, 5];
  public scheduleViews: View[] = ['Week', 'WorkWeek', 'Month',
'TimelineWeek', 'TimelineWorkWeek'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

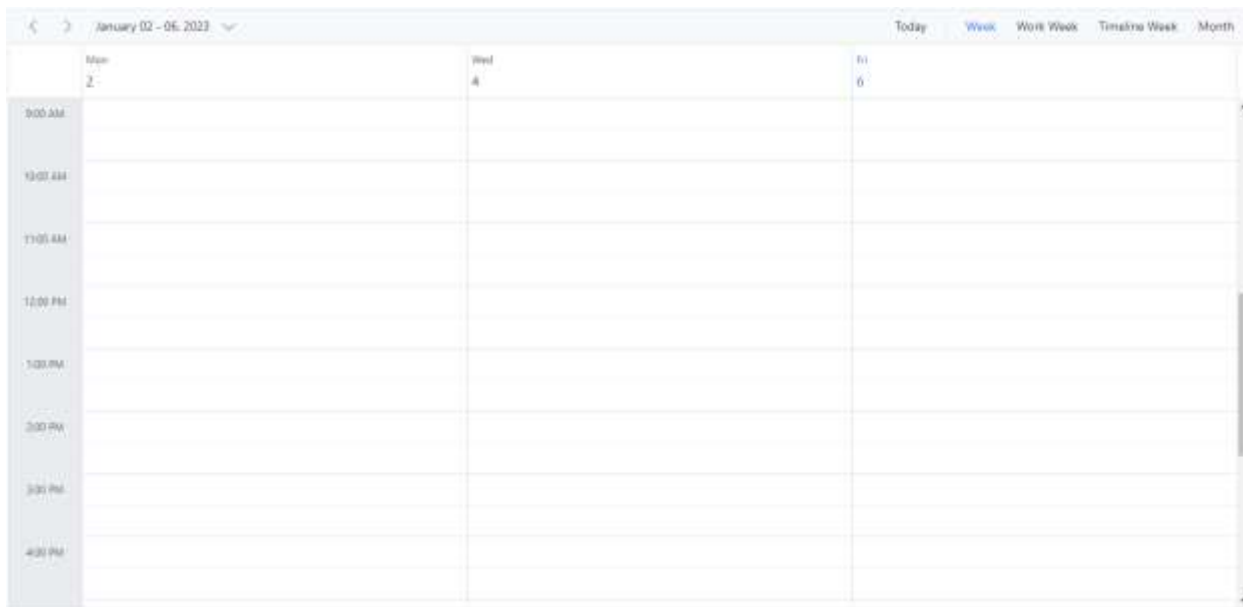
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Hiding weekend days

The `showWeekend` property is used to either show or hide the weekend days of a week and it is not applicable on Work week view (as non-working days are usually not displayed on work week view). By default, it is set to `true`. The days which are not a part of the working days collection of a Scheduler are usually considered as non-working or weekend days.

Here, the working days are defined as [1, 3, 4, 5] on Scheduler and therefore the remaining days (0, 2, 6 – Sunday, Tuesday and Saturday) are considered as non-working or weekend days and will be hidden from all the views when `showWeekend` property is set to `false`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, TimelineMonthService, View } from
 '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
})
export class AppComponent {
  // ...
}
```



```

standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [showWeekend]="showWeekend" [workDays]='workWeekDays'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
  })
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public workWeekDays: number[] = [1, 3, 4, 5];
  public showWeekend: boolean = false;
  public scheduleViews: View[] = ['Day', 'Week', 'TimelineMonth'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

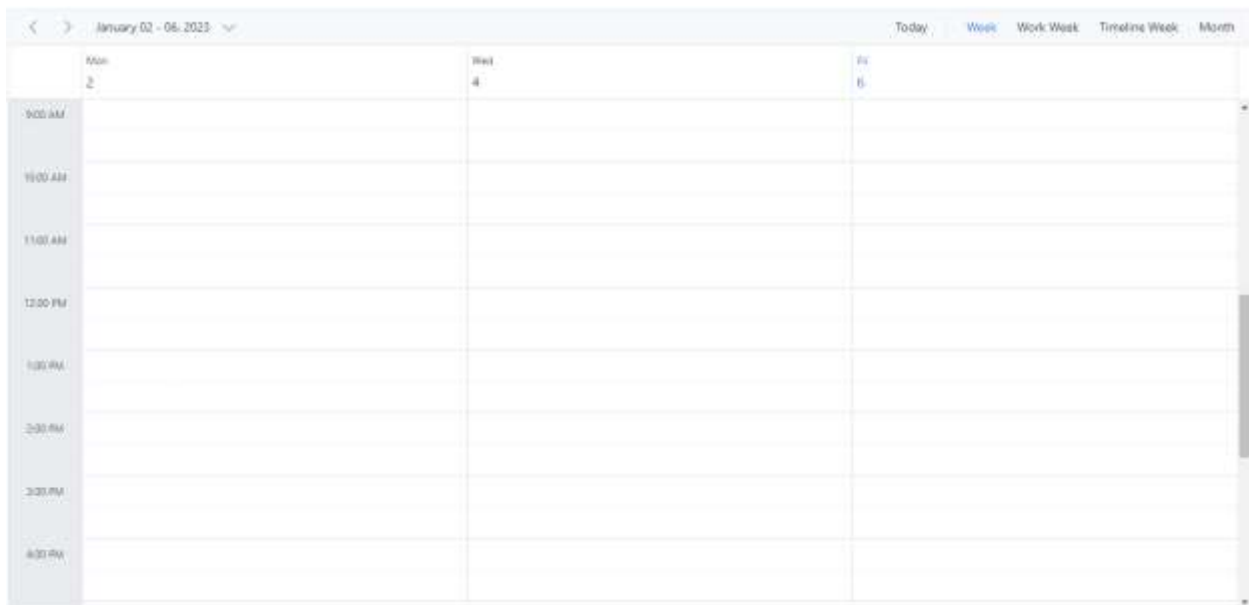
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Show week numbers

It is possible to show the week number count of a week in the header bar of the Scheduler by setting true to `showWeekNumber` property. By default, its default value is `false`. In Month view, the week numbers are displayed as a first column.

The `showWeekNumber` property is not applicable on Timeline views, as it has the equivalent `headerRows` property to handle such requirement with additional customizations.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, View } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' currentView='Month'
[views]="scheduleViews" [showWeekNumber]="showWeekNumber"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public workWeekDays: number[] = [1, 3, 4, 5];
  public showWeekNumber: boolean = true;
  public scheduleViews: View[] = ['Day', 'Week', 'Month'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

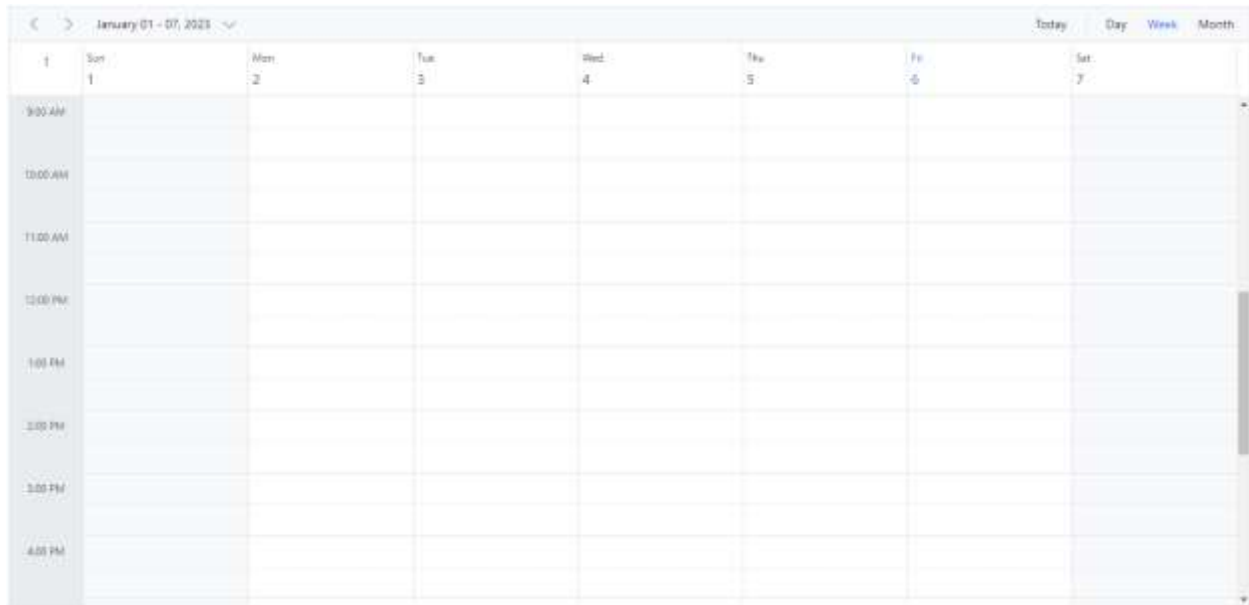
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Different options in showing week numbers

By default, week numbers are shown in the Scheduler based on the first day of the year. However, the week numbers can be determined based on the following criteria.

FirstDay – The first week of the year is calculated based on the first day of the year.

FirstFourDayWeek – The first week of the year begins from the first week with four or more days.

FirstFullWeek – The first week of the year begins when meeting the first day of the week (firstDayOfWeek) and the first day of the year.

For more details refer to [this link](#)

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, View, WeekRule } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
```

```

MonthService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' currentView='Month'
[views]="scheduleViews" [showWeekNumber]="showWeekNumber"
[weekRule]="weekRule" [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
))
export class AppComponent {
  public selectedDate: Date = new Date(2020, 1, 15);
  public workWeekDays: number[] = [1, 3, 4, 5];
  public showWeekNumber: boolean = true;
  public weekRule: WeekRule = 'FirstFourDayWeek';
  public scheduleViews: View[] = ['Day', 'Week', 'Month'];
  public eventSettings: EventSettingsModel = {
    dataSource: [
      {
        Id: 1,
        Subject: "Explosion of Betelgeuse Star",
        StartTime: new Date(2020, 1, 15, 9, 30),
        EndTime: new Date(2020, 1, 15, 11, 0)
      },
      {
        Id: 2,
        Subject: "Thule Air Crash Report",
        StartTime: new Date(2020, 1, 12, 12, 0),
        EndTime: new Date(2020, 1, 12, 14, 0)
      },
      {
        Id: 3,
        Subject: "Blue Moon Eclipse",
        StartTime: new Date(2020, 1, 13, 9, 30),
        EndTime: new Date(2020, 1, 13, 11, 0)
      },
      {
        Id: 4,
        Subject: "Meteor Showers in 2018",
        StartTime: new Date(2020, 1, 14, 13, 0),
        EndTime: new Date(2020, 1, 14, 14, 30)
      }
    ]
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Note: Enable the `showWeekNumber` property to configure the `weekRule` property. Also, the `weekRule` property depends on the value of the `firstDayOfWeek` property.

Set working hours

Working hours indicates the work hour limit within the Scheduler, which is visually highlighted with an active color on work cells. The working hours can be set on Scheduler using the `workHours` property which is of object type and includes the following sub-options,

- `highlight` – enables/disables the highlighting of work hours.
- `start` - sets the start time of the working/business hour of a day.
- `end` - sets the end time limit of the working/business hour of a day.

APP.COMPONENT.TS

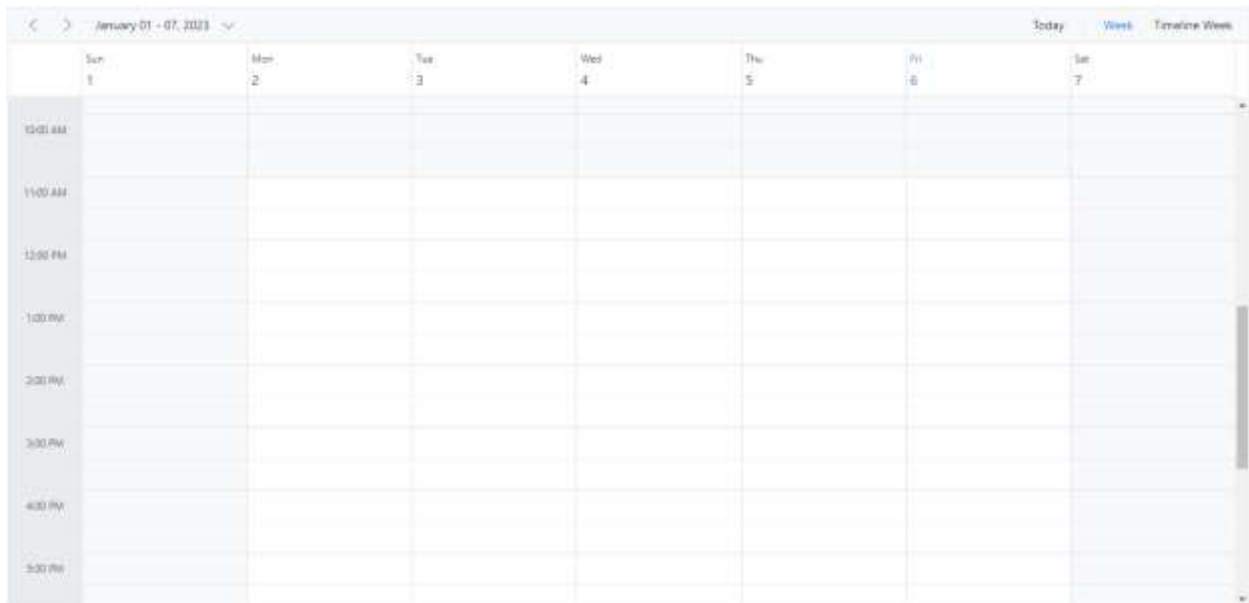
```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { MonthService, AgendaService, MonthAgendaService } from
 '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService, View,
 WorkHoursModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [workHours]="scheduleHours"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleHours: WorkHoursModel = { highlight: true, start: '11:00',
end: '20:00' };
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Scheduler displaying custom hours

It is possible to display the event Scheduler layout with specific time durations by hiding the unwanted hours. To do so, set the start and end hour for the Scheduler using the `startHour` and `endHour` properties respectively.

The following code example displays the Scheduler starting from the time range 7.00 AM to 6.00 PM and the remaining hours are hidden on the UI.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, View, WorkHoursModel } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
```

```

selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" startHour='07:00' endHour='18:00'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

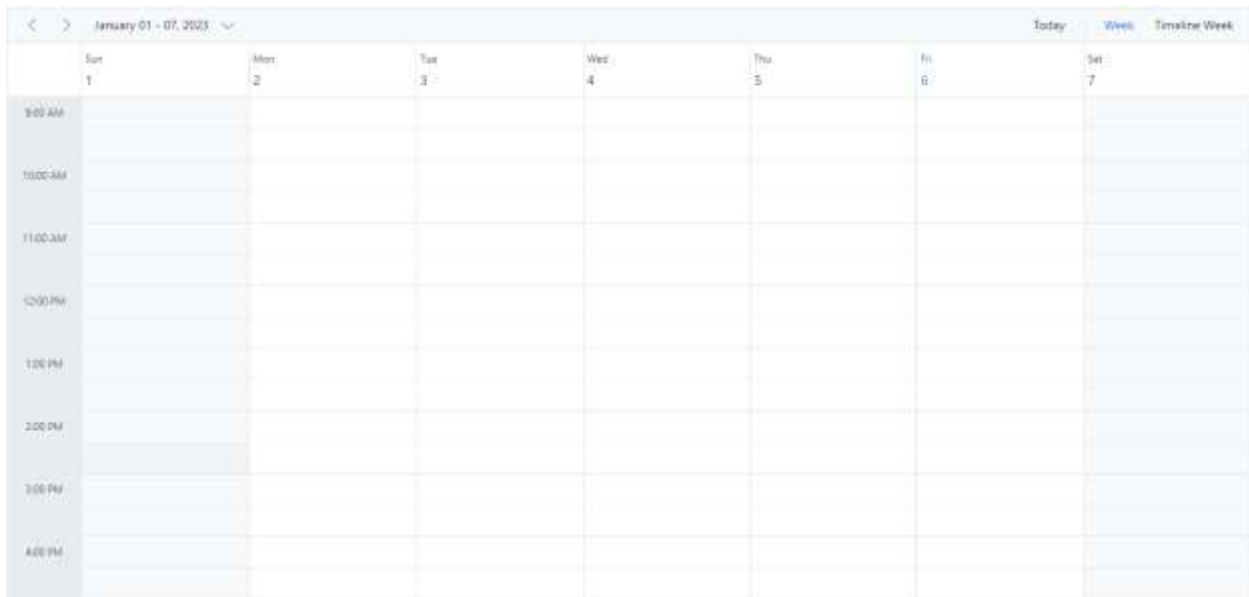
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Setting start day of the week

By default, Scheduler defaults to **Sunday** as its first day of a week. To change the Scheduler's start day of a week with different day, set the **firstDayOfWeek** property with the values ranging from 0 to 6.

Here, Sunday is always denoted as 0, Monday as 1 and so on.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WorkWeekService, MonthService, AgendaService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';

```

```

import { EventSettingsModel, WeekService, TimelineMonthService, View } from
 '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineMonthService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[views]="scheduleViews" [firstDayOfWeek]="weekFirstDay"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public weekFirstDay: number = 3;
  public scheduleViews: View[] = ['Week', 'TimelineMonth'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

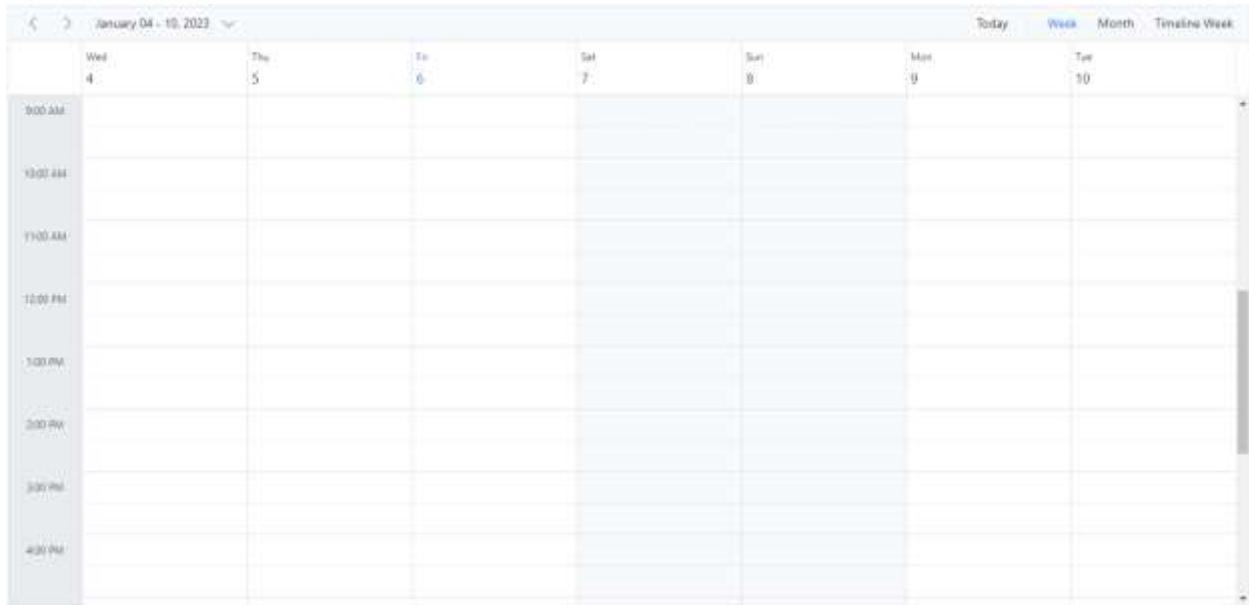
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Scroll to specific time and date

You can manually scroll to a specific time on Scheduler by making use of the `scrollTo` method as depicted in the following code example.

APP.COMPONENT.HTML

```
<table id="property" title="Properties" style="width: 100%">
  <tbody>
    <tr style="height: 50px">
      <td>
        <div> Scroll To
        </div>
      </td>
      <td>
        <div>
          <ejs-timepicker width='100px' [value]="scrollToHour"
format='HH:mm' (change)="onChange($event)"> </ejs-timepicker>
        </div>
      </td>
    </tr>
  </tbody>
</table>
<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild } from '@angular/core';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-calendars';
```

```
import { ScheduleComponent, EventSettingsModel, DayService, WeekService,
WorkWeekService, View, MonthService, AgendaService, MonthAgendaService } from
'@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    TimePickerModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public scrollToHour: Date = new Date(2018, 1, 15, 9);
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  onChange(args: ChangeEventArgs): void {
    this.scheduleObj!.scrollTo((args as any).text);
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

[How to scroll to current time on initial load](#)

There are scenarios where you may need to load the Scheduler displaying the system's current time on the currently visible view port area. In such cases, the Scheduler needs to be scrolled to a specific time based on the system's current time which is depicted in the following code example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, TimelineViewsService, View }
from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
```

```

imports: [
    ScheduleModule,
    ButtonModule
],
providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    TimelineViewsService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" (created)="onCreated($event)"
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
}))
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public scheduleViews: View[] = ['Day', 'Week', 'TimelineWorkWeek'];
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    onCreated(eventData: any): void {
        let currTime: Date = new Date();
        let hours: string = currTime.getHours() < 10 ? '0'
+currTime.getHours().toString() : currTime.getHours().toString();
        let minutes: string = currTime.getMinutes().toString();
        let time: string = hours + ':' + minutes;
        this.scheduleObj?.scrollTo(time);
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

See Also

- [To display the current time indicator](#)
- [To set different working hours dynamically](#)
- [To set different working hours for each resources](#)
- [To set different working days for each resources](#)

Cell customization in Angular Schedule component

The cells of the Scheduler can be easily customized either using the cell template or **RenderCell** event.

Setting cell dimensions in all views

The height and width of the Scheduler cells can be customized either to increase or reduce its size through the **cssClass** property, which overrides the default CSS applied on cells.

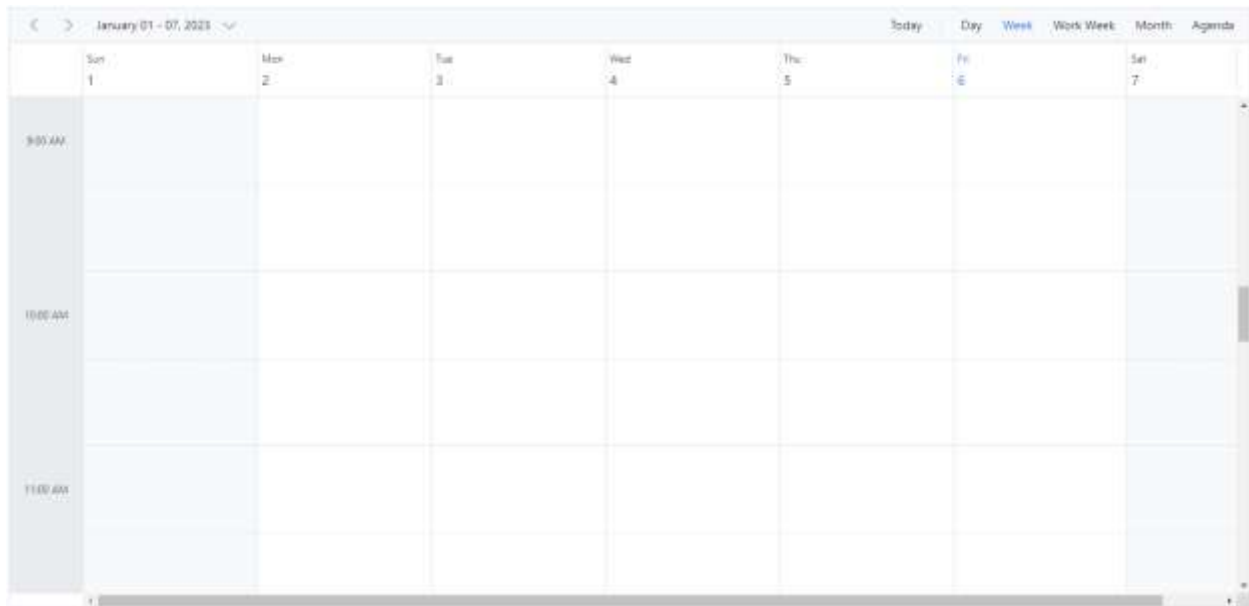
APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, ActionEventArgs, DayService,
    WeekService, WorkWeekService, MonthService, AgendaService,
    MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService,
    MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
cssClass='schedule-cell-dimension' [selectedDate]="selectedDate"
[eventSettings]="eventSettings"> </ejs-schedule>`,
  styles: [`
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
  width: 200px;
}
.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap table
td,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
  height: 100px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
  width: 200px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
  height: 200px;
}
`],
  encapsulation: ViewEncapsulation.None
})
```

```
export class AppComponent {
  @ViewChild('scheduleObj', { static: true })
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Check for cell availability

You can check whether the given time range slots are available for event creation or already occupied by other events using the `isSlotAvailable` method. In the following code example, if a specific time slot already contains an appointment, then no more appointments can be added to that cell.

Note: The `isSlotAvailable` is centered around verifying appointments within the present view's date range. Yet, it does not encompass an evaluation of availability for recurrence occurrences that fall beyond this particular date range.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, ActionEventArgs, DayService,
WeekService, WorkWeekService, MonthService, EventFieldsMapping, AgendaService
} from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
```

```

@Component({
  imports: [

    ScheduleModule,
    ButtonModule

  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService,
MonthService, AgendaService,
  MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(actionBegin)="onActionBegin($event)" >
    <e-views> <e-view option="Day"></e-view> <e-view option="Week"></e-view>
<e-view option="WorkWeek"></e-view> <e-view option="Month"></e-view></e-
views> </ejs-schedule>`,
  styles: [ `
#container {
  visibility: hidden;
}
#loader {
  color: #008cff;
  height: 40px;
  width: 30%;
  position: absolute;
  top: 45%;
  left: 45%;
}
.templatewrap {
  text-align: center;
  width: 100%;
}
.templatewrap img {
  width: 20px;
  height: 20px;
}
/* csslint ignore:start */
.schedule-cell-dimension.e-schedule .e-vertical-view .e-date-header-wrap
table col,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-content-wrap table
col {
  width: 200px;
}
.schedule-cell-dimension.e-schedule .e-vertical-view .e-time-cells-wrap table
th,
.schedule-cell-dimension.e-schedule .e-vertical-view .e-work-cells {
  height: 100px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells,
.schedule-cell-dimension.e-schedule .e-month-view .e-date-header-wrap table
col {
  width: 200px;
}
.schedule-cell-dimension.e-schedule .e-month-view .e-work-cells {
  height: 200px;
}

```

```

}
/* csslint ignore:end */
`],
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs): void {
    if (args.requestType === 'eventCreate' && (<Object[]>args.data).length >
0) {
      let eventData: { [key: string]: Object } = (<Object[]>args.data)[0] as {
[key: string]: Object };
      let eventField: EventFieldsMapping = this.scheduleObj?.eventFields as
EventFieldsMapping;
      let startDate: Date = eventData[eventField.startTime || ''] as Date;
      let endDate: Date = eventData[eventField.endTime || ''] as Date;
      args.cancel = !this.scheduleObj?.isSlotAvailable(startDate, endDate); }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing cells in all the views

It is possible to customize the appearance of the cells using both template options and `renderCell` event on all the views.

Using template

The `cellTemplate` option accepts the template string and is used to customize the cell background with specific images or appropriate text on the given date values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewEncapsulation } from '@angular/core';
import { View, DayService, WeekService, TimelineViewsService, MonthService,
WorkWeekService, AgendaService } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',

```

```

    providers: [DayService, WeekService, TimelineViewsService, MonthService,
    WorkWeekService, AgendaService,
    MonthAgendaService ],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [(currentView)]="currentView"
cssClass="schedule-cell-template">
    <ng-template #cellTemplate let-data>
        <div class="templatewrap" *ngIf="data.type == 'workCells'"
[innerHTML]="getWorkCellText(data.date)"></div>
        <div class="templatewrap" *ngIf="data.type == 'monthCells'"
[innerHTML]="getMonthCellText(data.date)"></div>
    </ng-template>
</ejs-schedule>`,
    styles: [`
.schedule-cell-template.e-schedule .e-month-view .e-work-cells {
    position: relative;
}
.templatewrap {
text-align: center;
}
/* In MONTH view the cell template is a SIBLING of event templates. So
which is necessary to set the parent position relative and the child position
absolute with 100% width */
.position: absolute;
width: 100%;
}
.templatewrap img {
width: 20px;
height: 20px;
}
`],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    public selectedDate: Date = new Date(2017, 11, 16);
    public currentView: View = 'Week';
    getMonthCellText(date: Date): string {
        if (date.getMonth() === 10 && date.getDate() === 23) {
            return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
        } else if (date.getMonth() === 11 && date.getDate() === 9) {
            return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
        } else if (date.getMonth() === 11 && date.getDate() === 13) {
            return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
        } else if (date.getMonth() === 11 && date.getDate() === 22) {
            return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
        } else if (date.getMonth() === 11 && date.getDate() === 24) {
            return '';
        } else if (date.getMonth() === 11 && date.getDate() === 25) {
            return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/christmas.svg" />';

```



```

    } else if (date.getMonth() === 0 && date.getDate() === 1) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg" />';
    } else if (date.getMonth() === 0 && date.getDate() === 14) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
}
getWorkCellText(date: Date): string {
    let weekEnds: number[] = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
        return "<img
src='https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg' />";
    }
    return '';
}
}
}

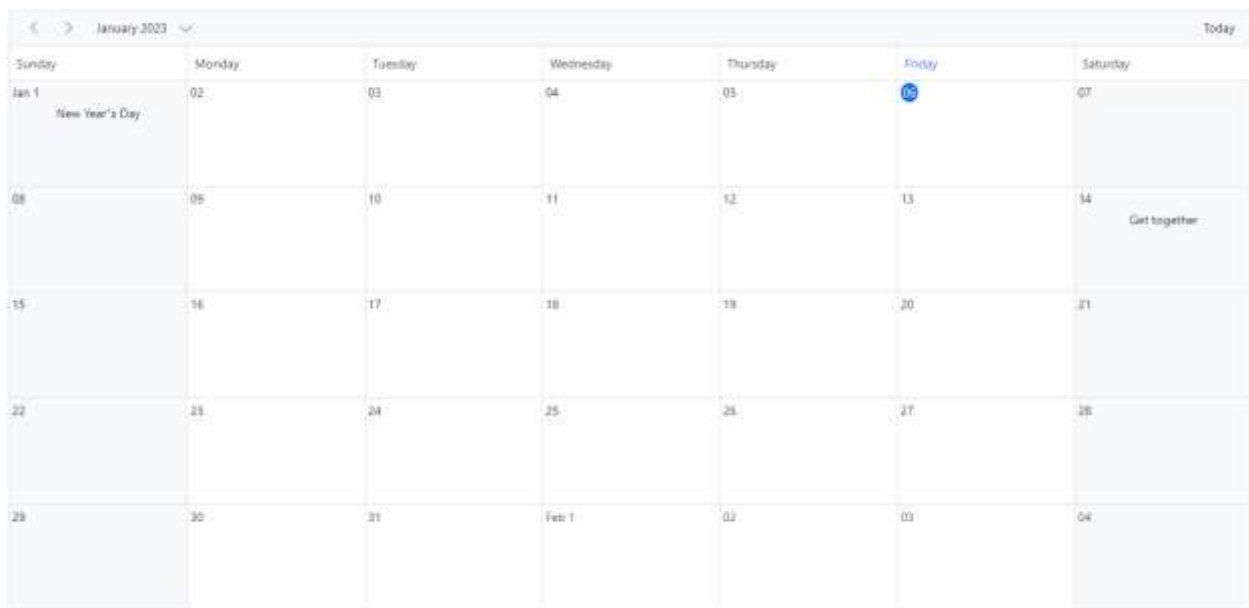
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Using renderCell event

An alternative to `cellTemplate` is the `renderCell` event, which can also be used to customize the cells with appropriate images or formatted text values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';

```

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { EventSettingsModel, RenderCellEventArgs } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from '../datasource';
import { createElement } from '@syncfusion/ej2-base';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' currentView='Month'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(renderCell)="onRenderCell($event)">
    <e-views> <e-view option="Day"></e-view> <e-view option="Week"></e-view>
<e-view option="Month"></e-view> </e-views> </ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 14);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  onRenderCell(args: RenderCellEventArgs): void {
    if (args.elementType == 'workCells' || args.elementType == 'monthCells')
    {
      let weekEnds: number[] = [0, 6];
      if (args.date && weekEnds.indexOf((args.date).getDay()) >= 0) {
        let ele: HTMLElement = createElement('div', {
          innerHTML: "<img
src='https://ej2.syncfusion.com/demos/src/schedule/images/newyear.svg' />",
          className: 'templatewrap'
        });
        (args.element).appendChild(ele);
      }
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can customize cells such as work cells, month cells, all-day cells, header cells, resource header cells using `renderCell` event by checking the `elementType` option within the event. You can check `elementType` with any of the following.

Element type	Description
----- -----	
dateHeader	triggers on header cell rendering.
monthDay	triggers on header cell in month view rendering.
resourceHeader	triggers on resource header cell rendering.
alldayCells	triggers on all day cell rendering.
emptyCells	triggers on empty cell rendering on header bar.
resourceGroupCells	triggers on rendering of work cells for parent resource.
workCells	triggers on work cell rendering.
monthCells	triggers on month cell rendering.
majorSlot	triggers on major time slot cell rendering.
minorSlot	triggers on minor time slot cell rendering.
weekNumberCell	triggers on cell displaying week number.

Customizing cell header in month view

The month header of each date cell in the month view can be customized using the `cellHeaderTemplate` option which accepts the string or `HTMLElement`. The corresponding date can be accessed with the template.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewEncapsulation } from '@angular/core';
import { Internationalization } from '@syncfusion/ej2-base';
import { View, } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
})
```

```

standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' cssClass="schedule-
cell-header-template">
  <ng-template #cellHeaderTemplate let-data>
    <div class="date-text">{{getDateHeaderText(data.date)}}</div>
  </ng-template>
  <e-views>
    <e-view option='Month'></e-view>
  </e-views>
</ejs-schedule>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public instance: Internationalization = new Internationalization();
  getDateHeaderText: Function = (value: Date) => {
    return this.instance.formatDate(value, { skeleton: "Ed" });
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing the minimum and maximum date values

Providing the `minDate` and `maxDate` property with some date values, allows the Scheduler to set the minimum and maximum date range. The Scheduler date that lies beyond this minimum and maximum date range will be in a disabled state so that the date navigation will be blocked beyond the specified date range.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { View, EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,

```

```

        AgendaService,
        MonthAgendaService],
standalone: true,
  selector: 'app-root',
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]='selectedDate' [minDate]='minDate' [maxDate]='maxDate'
[currentView]='currentView' [eventSettings]='eventSettings'></ejs-schedule>`
  })
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 17);
  public minDate: Date = new Date(2017, 4, 17);
  public maxDate: Date = new Date(2018, 8, 17);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public currentView: View = 'Month';
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

By default, the `minDate` property value is set to `new Date(1900, 0, 1)` and `maxDate` property value is set to `new Date(2099, 11, 31)`. The user can also set the customized `minDate` and `maxDate` property values.

Customizing the weekend cells background color

You can customize the background color of weekend cells by utilizing the [renderCell](#) event and checking the [elementType](#) option within the event.

```

`typescript
public onRenderCell(args: RenderCellEventArgs): void {
  if (args.elementType == "workCells") {
    if (args.date) {
      if (args.date.getDay() === 6) {
        (args.element as any).style.background = '#ffdea2';
      }
      if (args.date.getDay() === 0) {
        (args.element as any).style.background = '#ffdea2';
      }
    }
  }
}

```

And, the background color for weekend cells in the Month view through the [cssClass](#) property, which overrides the default CSS applied on cells.

`CSS

```
.schedule-cell-customization.e-schedule .e-month-view .e-work-cells:not(.e-work-days) {
background-color: #f08080;
}
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { Component, ViewEncapsulation } from '@angular/core';
import {
    EventSettingsModel, View, RenderCellEventArgs, DayService, WeekService,
    WorkWeekService,
    MonthService, AgendaService, ResizeService, DragAndDropService,
    ScheduleModule
} from '@syncfusion/ej2-angular-schedule';
import { defaultData } from './datasource';
@Component({
imports: [

    ScheduleModule
],
standalone: true,
selector: 'app-root',
providers: [View, DayService, WeekService, WorkWeekService, MonthService,
AgendaService, ResizeService, DragAndDropService],
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='650px' cssClass='schedule-
cell-customization' [selectedDate]="selectedDate"
[eventSettings]="eventSettings"
[(currentView)]="currentView" (renderCell)="onRenderCell($event)"></ejs-
schedule>`,
styles: [
`
.schedule-cell-customization.e-schedule .e-month-view .e-work-
cells:not(.e-work-days) {
background-color: #f08080;
}
`,
],
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
public selectedDate: Date = new Date(2021, 1, 15);
public eventSettings: EventSettingsModel = { dataSource: defaultData };
public currentView: View = 'Week';
public onRenderCell(args: RenderCellEventArgs): void {
if (args.elementType == "workCells") {
// To change the color of weekend columns in week view
if (args.date) {
if (args.date.getDay() === 6) {
(args.element as any).style.background = '#ffdea2';
}
```

```

    }
    if (args.date.getDay() === 0) {
      (args.element as any).style.background = '#ffdea2';
    }
  }
}
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to disable multiple cell and row selection in Schedule

By default, the `allowMultiCellSelection` and `allowMultiRowSelection` properties of the Schedule are set to `true`. So, the Schedule allows user to select multiple cells and rows. If the user want to disable this multiple cell and row selection. The user can disable this feature by setting up `false` to these properties.

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

State persistence in Angular Schedule component

State persistence allowed Scheduler to retain the `currentView`, `selectedDate` and Scroll position values in the [localStorage](#) for state maintenance even if the browser is refreshed or if you move to the next page within the browser. This action is handled through the `enablePersistence` property which is set to `false` by default. When it is set to `true`, `currentView`, `selectedDate` and Scroll position values of the scheduler component will be retained even after refreshing the page.

Note: Scheduler `id` is essential to set state persistence.

The following sample demonstrates how to set state persistence of the Scheduler component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel, TimeScaleModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],

```

```

providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule id="schedule" width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[enablePersistence]="enablePersistence"> </ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public enablePersistence: Boolean = true;
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Exporting in Angular Schedule component

The Scheduler supports exporting all its appointments both to an Excel or ICS extension file at client-side. It offers different client-side methods to export its appointments in an Excel or iCal format file. Let's look onto the ways on how to implement the exporting functionality in Scheduler.

Excel Exporting

The Scheduler allows you to export all its events into an Excel format file by using the `[exportToExcel]` client-side method. By default, it exports all the default fields of Scheduler mapped through `eventSettings` property.

Before you start with excel exporting functionality, you need to import and inject the `ExcelExport` module from the '@syncfusion/ej2-schedule' package using the `Inject` method of Scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';

```



```

import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
'@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
imports: [

        ScheduleModule,
        ButtonModule
    ],
providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService,
            ExcelExportService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" cssClass='schedule-excel-export'
[currentView]="currentView" (actionBegin)="onActionBegin($event)">
    <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
styleUrls: ['./index.css'],
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2019, 0, 10);
    public scheduleViews: View[] = ['Week'];
    public currentView: View = 'Week';
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
        if (args.requestType === 'toolbarItemRendering') {
            const exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
            };
            args.items?.push(exportItem);
        }
    }
    public onExportClick(): void {
        this.scheduleObj?.exportToExcel();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

	A	B	C	D	E	F	G	H	I
1	id	id	Subject	StartTime	EndTime	StartTimezone	EndTimezone	Location	Description
2	1		Explosion of Beigelrose Sta	1-8-2020 9:30 AM	1-8-2020 11:00 AM			Dallas	
3	2		Thule Air Crash Report	1-9-2020 12:00 PM	1-9-2020 2:00 PM			Texas	
4	3		Blue Moon Eclipse	1-10-2020 10:30 AM	1-10-2020 11:00 AM			Australia	
5	4		Meteor Showers in 2020	1-11-2020 1:00 PM	1-11-2020 2:30 PM			Canada	
6	5		Milky Way as Melting pot	1-12-2020 12:00 PM	1-12-2020 2:00 PM			Mexico	
7									

Exporting with custom fields

By default, Scheduler exports all the default event fields that are mapped to it through the `eventSettings` property. To limit the number of fields on the exported excel file, it provides an option to export only the custom fields of the event data. To export such custom fields alone, define the required fields through the `ExportOptions` interface and pass it as argument to the `exportToExcel` method as shown in the following example. For example: `['Id', 'Subject', 'StartTime', 'EndTime', 'Location']`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
'@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
imports: [

ScheduleModule,
ButtonModule
],
providers: [DayService,
WeekService,
WorkWeekService,
MonthService,
AgendaService,
MonthAgendaService,
ExcelExportService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate" cssClass='schedule-
excel-export' [eventSettings]="eventSettings" [currentView]="currentView"
(actionBegin)="onActionBegin($event)">
<e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
styleUrls: ['./index.css'],
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
```

```

@ViewChild('scheduleObj')
public scheduleObj?: ScheduleComponent;
public selectedDate: Date = new Date(2019, 0, 10);
public scheduleViews: View[] = ['Week'];
public currentView: View = 'Week';
public eventSettings: EventSettingsModel = { dataSource: scheduleData };
public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
        const exportItem: ItemModel = {
            align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
            excel-export',
            text: 'Excel Export', cssClass: 'e-excel-export', click:
            this.onExportClick.bind(this)
        };
        args.items?.push(exportItem);
    }
    public onExportClick(): void {
        const exportValues: ExportOptions = { fields: ['Id', 'Subject',
        'StartTime', 'EndTime', 'Location'] };
        this.scheduleObj?.exportToExcel(exportValues);
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

	A	B	C	D	E
1	Id	Subject	StartTime	EndTime	Location
2	1	Explosion of Betelgeuse Sta	1-8-2020 9:30 AM	1-8-2020 11:00 AM	Dallas
3	2	Thule Air Crash Report	1-9-2020 12:00 PM	1-9-2020 2:00 PM	Texas
4	3	Blue Moon Eclipse	1-10-2020 10:30 AM	1-10-2020 11:00 AM	Australia
5	4	Meteor Showers in 2020	1-11-2020 1:00 PM	1-11-2020 2:30 PM	Canada
6	5	Milky Way as Melting pot	1-12-2020 12:00 PM	1-12-2020 2:00 PM	Mexico
7					

Exporting individual occurrences of a recurring series

By default, the Scheduler exports recurring events as a single data by exporting only its parent record into the excel file. If you want to export each individual occurrences of a recurring series appointment as separate records in an Excel file, define the `includeOccurrences` option as `true` through the `ExportOptions` interface and pass it as argument to the `exportToExcel` method. By default, the `includeOccurrences` option is set to `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';

```

```

import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
 '@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ExcelExportService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate" cssClass='schedule-
excel-export' [eventSettings]="eventSettings" [currentView]="currentView"
(actionBegin)="onActionBegin($event)">
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2019, 0, 10);
  public scheduleViews: View[] = ['Week'];
  public currentView: View = 'Week';
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
      const exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
      };
      args.items?.push(exportItem);
    }
  }
  public onExportClick(): void {
    const exportValues: ExportOptions = { includeOccurrences: true };
    this.scheduleObj?.exportToExcel(exportValues);
  }
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Exporting custom event data

By default, the whole event collection bound to the Scheduler gets exported as an excel file. To export only specific events of Scheduler or some custom event collection, you need to pass those custom data collection as a parameter to the `exportToExcel` method as shown in this following example, through the `customData` option of `ExportOptions` interface.

By default, the event data are taken from Scheduler `dataSource`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
'@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ExcelExportService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" cssClass='schedule-excel-export'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[currentView]="currentView" (actionBegin)="onActionBegin($event)">
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
```

```

public selectedDate: Date = new Date(2019, 0, 10);
public scheduleViews: View[] = ['Week'];
public currentView: View = 'Week';
public eventSettings: EventSettingsModel = { dataSource: scheduleData };
public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
        const exportItem: ItemModel = {
            align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
            excel-export',
            text: 'Excel Export', cssClass: 'e-excel-export', click:
            this.onExportClick.bind(this)
        };
        args.items?.push(exportItem);
    }
}
public onExportClick(): void {
    const exportValues: ExportOptions = {
        customData: [{
            Id: 1,
            Subject: 'Explosion of Betelgeuse Star',
            Location: 'Space Centre USA',
            StartTime: new Date(2019, 0, 6, 9, 30),
            EndTime: new Date(2019, 0, 6, 11, 0),
            CategoryColor: '#1aaa55'
        }, {
            Id: 2,
            Subject: 'Thule Air Crash Report',
            Location: 'Newyork City',
            StartTime: new Date(2019, 0, 7, 12, 0),
            EndTime: new Date(2019, 0, 7, 14, 0),
            CategoryColor: '#357cd2'
        }]
    };
    this.scheduleObj?.exportToExcel(exportValues);
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Customizing column header with custom fields exporting

Using fields property, we can only export the defined fields into excel without customizing the header. Now we can provide the alternate support to customize the header of custom fields exporting using the `fieldsInfo` option through the `ExportFieldInfo` interface and pass it as an argument to the `exportToExcel` method as shown in the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'

```

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import {
    ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
    ToolbarActionArgs, ExportOptions,
    ExportFieldInfo, ExcelExportService
} from '@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from '../datasource';
@Component({
    imports: [

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService,
        ExcelExportService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate" cssClass='schedule-
excel-export' [eventSettings]="eventSettings" [currentView]="currentView"
(actionBegin)="onActionBegin($event)">
    <e-views><e-view option='Week'></e-view></e-views></ejs-schedule>`,
    styleUrls: ['./index.css'],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2019, 0, 10);
    public scheduleViews: View[] = ['Week'];
    public currentView: View = 'Week';
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
        if (args.requestType === 'toolbarItemRendering') {
            const exportItem: ItemModel = {
                align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
                text: 'Excel Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
            };
            args.items?.push(exportItem);
        }
    }
    public onExportClick(): void {
        const customFields: ExportFieldInfo[] = [
            { name: 'Subject', text: 'Summary' },
            { name: 'StartTime', text: 'First Date' },

```

```

        { name: 'EndTime', text: 'Last Date' },
        { name: 'Location', text: 'Place' },
        { name: 'OwnerId', text: 'Owners' }
    ];
    const exportValues: ExportOptions = { fieldsInfo: customFields };
    this.scheduleObj?.exportToExcel(exportValues);
}
}

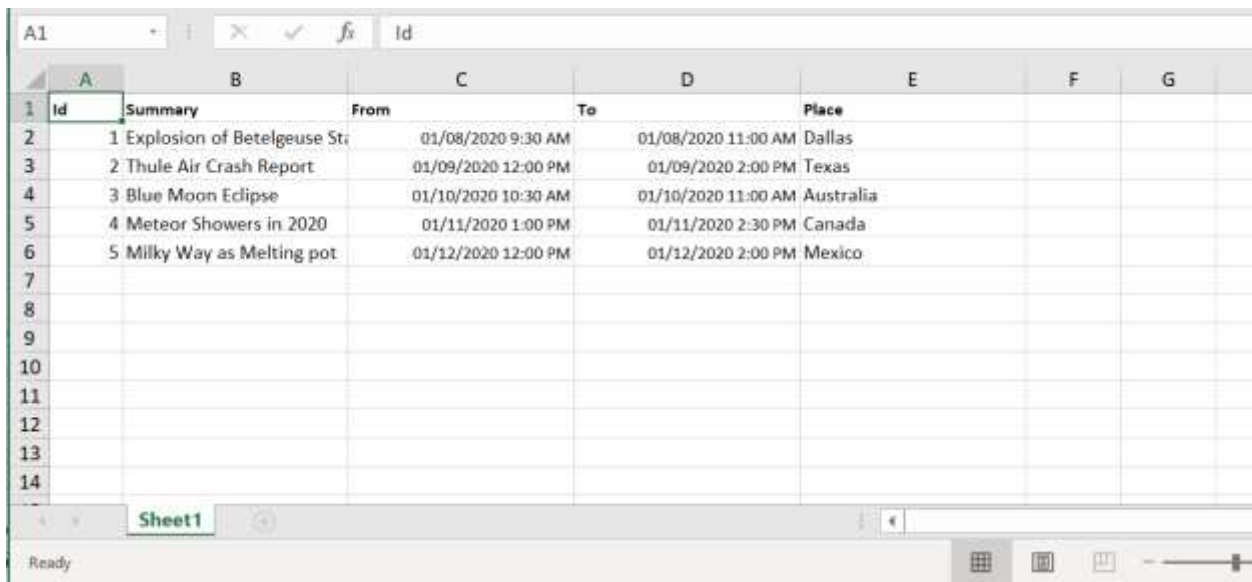
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



	A	B	C	D	E	F	G
	Id	Summary	From	To	Place		
1	1	Explosion of Betelgeuse Str	01/08/2020 9:30 AM	01/08/2020 11:00 AM	Dallas		
2	2	Thule Air Crash Report	01/09/2020 12:00 PM	01/09/2020 2:00 PM	Texas		
3	3	Blue Moon Eclipse	01/10/2020 10:30 AM	01/10/2020 11:00 AM	Australia		
4	4	Meteor Showers in 2020	01/11/2020 1:00 PM	01/11/2020 2:30 PM	Canada		
5	5	Milky Way as Melting pot	01/12/2020 12:00 PM	01/12/2020 2:00 PM	Mexico		
6							
7							
8							
9							
10							
11							
12							
13							
14							

Export with custom file name

By default, the Scheduler allows you to download the exported Excel file with a name `Schedule.xlsx`. It also provides an option to export the excel file with a custom file name, by defining the desired `fileName` through the `ExportOptions` interface and passing it as an argument to the `exportToExcel` method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from '@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';

```



```

import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ExcelExportService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" cssClass='schedule-excel-export'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[currentView]="currentView" (actionBegin)="onActionBegin($event)">
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2019, 0, 10);
  public scheduleViews: View[] = ['Week'];
  public currentView: View = 'Week';
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
      const exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
        text: 'Excel Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
      };
      args.items?.push(exportItem);
    }
  }
  public onExportClick(): void {
    const exportValues: ExportOptions = { fileName: "SchedulerData" };
    this.scheduleObj?.exportToExcel(exportValues);
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Excel file formats

By default, the Scheduler exports event data to an excel file in the .xlsx format. You can also export the Scheduler data in either of the file type such as .xlsx or csv formats, by defining the `exportType` option as either `csv` or `xlsx` through the `ExportOptions` interface. By default, the `exportType` is set to `xlsx`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
 '@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ExcelExportService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" cssClass='schedule-excel-export'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[currentView]="currentView" (actionBegin)="onActionBegin($event)">
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2019, 0, 10);
  public scheduleViews: View[] = ['Week'];
  public currentView: View = 'Week';
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
      const exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
```

```

        text: 'Excel Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
    };
    args.items?.push(exportItem);
}
}
public onExportClick(): void {
    const exportValues: ExportOptions = { exportType: "csv" };
    this.scheduleObj?.exportToExcel(exportValues);
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Custom separator in CSV

The Scheduler exports the event data to CSV format with `,` as separator. You can change separator by setting [separator](#) property in [ExportOptions](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, ActionEventArgs,
ToolbarActionArgs, ExportOptions, ExcelExportService } from
'@syncfusion/ej2-angular-schedule';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    ExcelExportService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" cssClass='schedule-excel-export'

```

```
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[currentView]="currentView" (actionBegin)="onActionBegin($event)">
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`,
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2019, 0, 10);
  public scheduleViews: View[] = ['Week'];
  public currentView: View = 'Week';
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs & ToolbarEventArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
      const exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
excel-export',
        text: 'CSV-Export', cssClass: 'e-excel-export', click:
this.onExportClick.bind(this)
      };
      args.items?.push(exportItem);
    }
  }
  public onExportClick(): void {
    const exportValues: ExportOptions = { exportType: 'csv', separator: ';' };
    this.scheduleObj?.exportToExcel(exportValues);
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Exporting calendar events as ICS file

You can export the Scheduler events to a calendar (.ics) file format, and open it on any of the other default calendars such as Google or Outlook. To export the events of Scheduler to an ICS file, you need to first import the `ICalendarExport` module from `@syncfusion/ej2-schedule` package and then inject it using the `Schedule.Inject(ICalendarExport)` method.

The following code example shows how the Scheduler events are exported to a calendar (.ics) file by making use of the `exportToCalendar` public method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
```

```

import { ScheduleComponent, EventSettingsModel, View, DayService,
DragAndDropService,
    WeekService, WorkWeekService, MonthService, AgendaService,
ICalendarExportService, ICalendarImportService, ResizeService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
imports: [

    ScheduleModule,
    ButtonModule,
    UploaderModule

],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, ResizeService,
    ICalendarExportService, ICalendarImportService, DragAndDropService,
MonthAgendaService],
    // specifies the template string for the Schedule component
    template: `<button ejs-button id="ics-export" type="button"
(click)="onExportClick()">Export</button>
<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [currentView]="currentView" >
</ejs-schedule>`
})
export class AppComponent {
    @ViewChild('scheduleObj', { static: true })
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month',
'Agenda'];
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public currentView: View = 'Week';
    public onExportClick(): void {
        this.scheduleObj?.exportToICalendar();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Exporting calendar with custom file name

By default, the calendar is exported with a file name **Calendar.ics**. To change this file name on export, pass the custom string value as **fileName** to the method argument so as to get the file downloaded with this provided name.

The following example downloads the iCal file with a name **ScheduleEvents.ics**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, DayService,
DragAndDropService,
    WeekService, WorkWeekService, MonthService, AgendaService,
ICalendarExportService, ICalendarImportService, ResizeService,
MonthAgendaService} from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
imports: [

    ScheduleModule,
    ButtonModule,
    UploaderModule
],
standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, ResizeService,
    ICalendarExportService, ICalendarImportService, DragAndDropService,
MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<button ej2-button id="ics-export" type="button"
(click)="onExportClick()">Export</button>
<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [currentView]="currentView" >
  <e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month',
'Agenda'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public currentView: View = 'Week';
  public onExportClick=(): void =>{
    this.scheduleObj?.exportToICalendar('ScheduleEvents');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Import events from other calendars

The events from external calendars (ICS files) can be imported into Scheduler by using the `importICalendar` method. This method accepts the `blob object` of an .ics file to be imported, as a mandatory argument.

To import an ICS file containing events into Scheduler, you need to first import the `ICalendarImport` module from `@syncfusion/ej2-schedule` package and then inject it using the `Schedule.Inject(ICalendarImport)` method.

The following example shows how to import an ICS file into Scheduler, using the `importICalendar` method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View, DayService,
DragAndDropService,
WeekService, WorkWeekService, MonthService, AgendaService,
ICalendarExportService, ICalendarImportService, ResizeService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
import { SelectedEventArgs } from '@syncfusion/ej2-inputs';
@Component({
imports: [

ScheduleModule,
ButtonModule,
UploaderModule

],
standalone: true,
selector: 'app-root',
providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, ResizeService,
ICalendarExportService, ICalendarImportService, DragAndDropService,
MonthAgendaService],
// specifies the template string for the Schedule component
template: `<ejs-uploader id='ics-import' cssClass='calendar-import'
[multiple]='multiple' [buttons]='buttons'
[showFileList]='showFileList' allowedExtensions='.ics'
(selected)='onSelected($event)'></ejs-uploader>
<ejs-schedule #scheduleObj width='100%' height='550px'
[views]="scheduleViews" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" [currentView]="currentView" >
<e-views><e-view option='Week' ></e-view></e-views></ejs-schedule>`
})
export class AppComponent {
@ViewChild('scheduleObj')
public scheduleObj?: ScheduleComponent;
public selectedDate: Date = new Date(2018, 1, 15);
public scheduleViews: View[] = ['Day', 'Week', 'WorkWeek', 'Month',
'Agenda'];
```

```

public eventSettings: EventSettingsModel = { dataSource: scheduleData };
public currentView: View = 'Week';
public showFileList: Boolean = false;
public multiple: Boolean = false;
public buttons: Object = { browse: 'Choose file' };
public onSelect(edargs: SelectedEventArgs): void {
    this.scheduleObj?.importICalendar(((HTMLInputElement>args.event.target)
as any).files[0]));
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

How to print the Scheduler element

The Scheduler allows you to print the Scheduler element by using the `print` client-side method. The `print` method works in two ways. You can find it below.

- Using `print` method without options.
- Using a `print` method with options.

To print the Schedule, you need to import the `PrintService` module from the `@syncfusion/ej2-angular-schedule` package and then inject it using `providers:[PrintService]`.

Using `print` method without options

You can print the Schedule element with the current view by using the `print` method without passing the options. The following example shows how to print the Scheduler using the `print` method without passing options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { UploaderModule } from '@syncfusion/ej2-angular-inputs';
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { ScheduleComponent, EventSettingsModel, ActionEventArgs,
ToolbarActionArgs, View, PrintService,
DayService, WeekService, WorkWeekService, MonthService, AgendaService,
MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { scheduleData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        ButtonModule,
        UploaderModule
    ],
})

```



```

    ],
    standalone: true,
    selector: 'app-root',
    providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService, PrintService, MonthAgendaService],
    // specifies the template string for the Schedule component
    template: `<ejs-schedule #scheduleObj id="schedule" cssClass='schedule-
    print' width='100%' height='550px' [selectedDate]="selectedDate"
    [eventSettings]="eventSettings" (actionBegin)="onActionBegin($event)"></ejs-
    schedule>`
  })
  export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
      if (args.requestType === 'toolbarItemRendering') {
        const exportItem: ItemModel = {
          align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
          print',
          text: 'Print', cssClass: 'e-print', click:
        this.onPrintIconClick.bind(this)
        };
        args.items?.push(exportItem);
      }
    }
    public onPrintIconClick(): void {
      this.scheduleObj?.print();
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Using a print method with options

You can print the Schedule element based on your needs using the `print` method by passing the print options used in this example with its values. The following example shows how to print the Scheduler using the `print` method by passing the options.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { UploaderModule } from '@syncfusion/ej2-angular-inputs'
import { ItemModel } from '@syncfusion/ej2-angular-navigations';
import { ScheduleComponent, EventSettingsModel, ActionEventArgs,
    ToolbarActionArgs, View, PrintService,

```

```

    DayService, WeekService, WorkWeekService, MonthService, AgendaService,
    ScheduleModel, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { scheduleData } from '../datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule,
    UploaderModule

  ],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService, PrintService, MonthAgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj id="schedule" cssClass='schedule-
    print' width='100%' height='550px' [selectedDate]="selectedDate"
    [eventSettings]="eventSettings" (actionBegin)="onActionBegin($event)"></ejs-
    schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  public onActionBegin(args: ActionEventArgs & ToolbarActionArgs): void {
    if (args.requestType === 'toolbarItemRendering') {
      const exportItem: ItemModel = {
        align: 'Right', showTextOn: 'Both', prefixIcon: 'e-icon-schedule-
        print',
        text: 'Print', cssClass: 'e-print', click:
        this.onPrintIconClick.bind(this)
      };
      args.items!.push(exportItem);
    }
  }
  public onPrintIconClick(): void {
    let printModel: ScheduleModel = {
      agendaDaysCount: 14,
      cssClass: 'e-print-schedule',
      currentView: this.scheduleObj?.currentView,
      dateFormat: 'dd-MMM-yyyy',
      enableRtl: false,
      endHour: '18:00',
      firstDayOfWeek: 1,
      firstMonthOfYear: 0,
      group: {},
      height: 'auto',
      locale: this.scheduleObj?.locale,
      maxDate: this.scheduleObj?.selectedDate,
      minDate: this.scheduleObj?.getCurrentViewDates()[0],
      readonly: true,
      resources: [],
      rowAutoHeight: false,
      selectedDate: new Date(),
      showHeaderBar: false,

```

```

        showTimeIndicator: false,
        showWeekNumber: false,
        showWeekend: false,
        startHour: '06:00',
        timeFormat: 'HH',
        timeScale: { enable: true },
        width: 'auto',
        workDays: [1, 2, 3, 4, 5],
        workHours: { highlight: true, start: '10:00', end: '20:00' }
    };
    this.scheduleObj?.print(printModel);
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Context menu in Angular Schedule component

You can display context menu on work cells and appointments of Scheduler by making use of the **ContextMenu** control manually from the application end. In the following code example, context menu control is being added from sample end and set its target as **Scheduler**.

On Scheduler cells, you can display the menu items such as **New Event**, **New Recurring Event** and **Today** option. For appointments, you can display its related options such as **Edit Event** and **Delete Event**. The default event window can be opened for appointment creation and editing using the **openEditor** method of Scheduler.

The deletion of appointments can be done by using the **deleteEvent** public method. Also, the **selectedDate** property can be used to navigate between different dates.

You can also display custom menu options on Scheduler cells and appointments. Context menu will open on tap-hold in responsive mode.

APP.COMPONENT.HTML

```

<ejs-schedule #scheduleObj height='550px' [selectedDate]="selectedDate"
[allowResizing]="allowResizing"
[allowDragAndDrop]="allowDragDrop" [eventSettings]="eventSettings">
</ejs-schedule>
<ejs-contextmenu #menuObj cssClass='schedule-context-menu' target='.e-
schedule' [items]='menuItems' (beforeOpen)='onContextMenuBeforeOpen($event)'
(select)='onMenuItemSelect($event)'></ejs-contextmenu>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { ContextMenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { extend, closest, isNullOrUndefined, remove, removeClass } from
 '@syncfusion/ej2-base';
import { DataManager, Query } from '@syncfusion/ej2-data';
import {
    EventSettingsModel, DayService, WeekService, WorkWeekService,
    MonthService, AgendaService, ScheduleComponent, CellClickEventArgs,
    MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
import { ContextMenuComponent, MenuItemModel, BeforeOpenCloseMenuEventArgs,
    MenuEventArgs } from '@syncfusion/ej2-angular-navigations';
import { scheduleData } from './datasource';
@Component({
    imports: [

        ScheduleModule,
        ButtonModule,
        ContextMenuModule
    ],
    standalone: true,
    selector: 'app-root',
    templateUrl: './app.component.html',
    providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService, MonthAgendaService],
    styleUrls: ['./index.css'],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    @ViewChild('menuObj')
    public menuObj?: ContextMenuComponent;
    public allowResizing: Boolean = false;
    public allowDragDrop: Boolean = false;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource:
    <Object[]>extend([], scheduleData, undefined, true) };
    public selectedTarget?: Element;
    public menuItems: MenuItemModel[] = [
        {
            text: 'New Event',
            iconCss: 'e-icons new',
            id: 'Add'
        }, {
            text: 'New Recurring Event',
            iconCss: 'e-icons recurrence',
            id: 'AddRecurrence'
        }, {
            text: 'Today',
            iconCss: 'e-icons today',
            id: 'Today'
        }, {

```

```

        text: 'Edit Event',
        iconCss: 'e-icons edit',
        id: 'Save'
    }, {
        text: 'Edit Event',
        id: 'EditRecurrenceEvent',
        iconCss: 'e-icons edit',
        items: [{
            text: 'Edit Occurrence',
            id: 'EditOccurrence'
        }, {
            text: 'Edit Series',
            id: 'EditSeries'
        }]
    }, {
        text: 'Delete Event',
        iconCss: 'e-icons delete',
        id: 'Delete'
    }, {
        text: 'Delete Event',
        id: 'DeleteRecurrenceEvent',
        iconCss: 'e-icons delete',
        items: [{
            text: 'Delete Occurrence',
            id: 'DeleteOccurrence'
        }, {
            text: 'Delete Series',
            id: 'DeleteSeries'
        }]
    }
];

onContextMenuBeforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
    const newEventElement: HTMLElement = document.querySelector('.e-new-event') as HTMLElement;
    if (newEventElement) {
        remove(newEventElement);
        removeClass([document.querySelector('.e-selected-cell') as Element], 'e-selected-cell');
    }
    this.scheduleObj?.closeQuickInfoPopup();
    const targetElement: HTMLElement = <HTMLElement>args.event.target;
    if (closest(targetElement, '.e-contextmenu')) {
        return;
    }
    this.selectedTarget = closest(targetElement, '.e-appointment, .e-work-cells, ' +
        '.e-vertical-view .e-date-header-wrap .e-all-day-cells, .e-vertical-view .e-date-header-wrap .e-header-cells');
    if (isNullOrUndefined(this.selectedTarget)) {
        args.cancel = true;
        return;
    }
    if (this.selectedTarget.classList.contains('e-appointment')) {
        const eventObj: { [key: string]: Object } = <{ [key: string]: Object }>this.scheduleObj?.getEventDetails(this.selectedTarget);
        if (eventObj['RecurrenceRule']) {

```

```

        this.menuObj?.showItems(['EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
        this.menuObj?.hideItems(['Add', 'AddRecurrence', 'Today',
'Save', 'Delete'], true);
    } else {
        this.menuObj?.showItems(['Save', 'Delete'], true);
        this.menuObj?.hideItems(['Add', 'AddRecurrence', 'Today',
'EditRecurrenceEvent', 'DeleteRecurrenceEvent'], true);
    }
    return;
}
this.menuObj?.hideItems(['Save', 'Delete', 'EditRecurrenceEvent',
'DeleteRecurrenceEvent'], true);
this.menuObj?.showItems(['Add', 'AddRecurrence', 'Today'], true);
}
onMenuItemSelect(args: MenuEventArgs): void {
    const selectedItem: string | undefined = args.item.id;
    let eventObj: { [key: string]: Object } = {};
    if (this.selectedTarget && this.selectedTarget.classList.contains('e-
appointment')) {
        eventObj = <{ [key: string]: Object
}>this.scheduleObj?.getEventDetails(this.selectedTarget);
    }
    switch (selectedMenuItem) {
        case 'Today':
            this.scheduleObj!.selectedDate = new Date();
            break;
        case 'Add':
        case 'AddRecurrence':
            const selectedCells: Element[] =
this.scheduleObj!.getSelectedElements();
            const activeCellsData: CellClickEventArgs =
this.scheduleObj!.getCellDetails(selectedCells.length > 0 ? selectedCells :
this.selectedTarget as any);
            if (selectedMenuItem === 'Add') {
                this.scheduleObj?.openEditor(activeCellsData, 'Add');
            } else {
                this.scheduleObj?.openEditor(activeCellsData, 'Add',
undefined, 1);
            }
            break;
        case 'Save':
        case 'EditOccurrence':
        case 'EditSeries':
            if (selectedMenuItem === 'EditSeries') {
                eventObj = <{ [key: string]: Object }>new
DataManager(this.scheduleObj?.eventsData).executeLocal(new Query().
                where(this.scheduleObj?.eventFields.id as any,
'equal', eventObj[this.scheduleObj?.eventFields.recurrenceID as any] as
string | number))[0];
            }
            this.scheduleObj?.openEditor(eventObj, selectedMenuItem);
            break;
        case 'Delete':
            this.scheduleObj?.deleteEvent(eventObj);
            break;
        case 'DeleteOccurrence':

```

```

        case 'DeleteSeries':
            this.scheduleObj?.deleteEvent(eventObj, selectedMenuItem);
            break;
    }
}
}

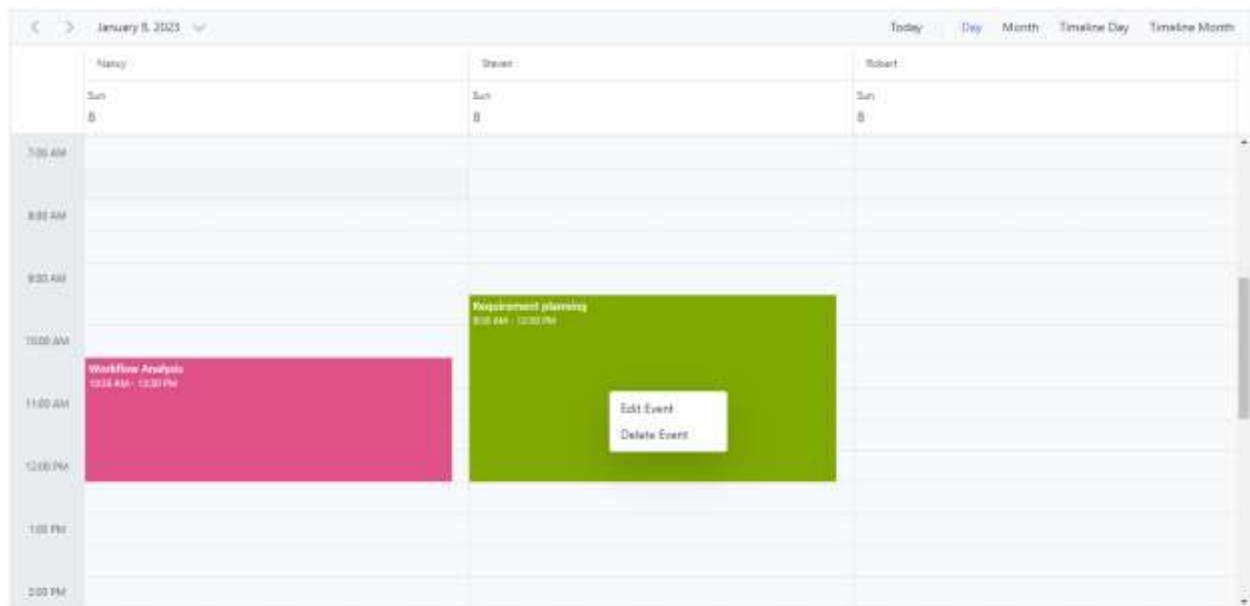
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Dimensions in Angular Schedule component

The Scheduler dimensions refers to both height and width of the entire layout and it accepts 3 types of values.

- auto
- pixel
- percentage

Auto Height and Width

When height and width of the Scheduler 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 Scheduler, its width/height will be the sum of its children. By default, Scheduler is assigned with **auto** values for both height and width properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='auto' height='auto'
[selectedDate]='selectedDate' [eventSettings]='eventSettings'></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Height and Width in pixel

The Scheduler height and width will be rendered exactly as per the given pixel values. It accepts both string and number values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'

```



```
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='650px' height='550px'
[selectedDate]='selectedDate' [eventSettings]='eventSettings'></ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData
  };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Height and Width in percentage

When height and width of the Scheduler are given as percentage, it will make the Scheduler as wide as the parent container.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { DayService, MonthService, AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule';
import { Component } from '@angular/core';
import { EventSettingsModel, WeekService, WorkWeekService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
```

```

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='100%'
[selectedDate]='selectedDate' [eventSettings]='eventSettings'></ejs-
schedule>`
  })
  export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = {
      dataSource: scheduleData
    };
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

See Also

- [How to Change Scheduler Cell Dimensions](#)

Recurrence editor in Angular Schedule component

The Recurrence editor is integrated into Scheduler editor window by default, to process the recurrence rule generation for events. Apart from this, it can also be used as an individual component referring from the Scheduler repository to work with the recurrence related processes.

All the valid recurrence rule string mentioned in the [iCalendar](#) specifications are applicable to use with the recurrence editor.

Customizing the repeat type option in editor

By default, there are 5 types of repeat options available in recurrence editor such as,

- Never
- Daily
- Weekly

- Monthly
- Yearly

It is possible to customize the recurrence editor to display only the specific repeat options such as **Daily** and **Weekly** options alone by setting the appropriate **frequencies** option.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, PopupOpenEventArgs, ScheduleComponent } from
'@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]='selectedDate'
[eventSettings]='eventSettings' (popupOpen)='onPopupOpen($event)'>
</ejs-schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  onPopupOpen(args: PopupOpenEventArgs): void {
    if (args.type == 'Editor') {
      (<any>this.scheduleObj!.eventWindow).recurrenceEditor.frequencies =
['daily', 'weekly'];
    }
  }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

The other properties available in recurrence editor are tabulated below,

Properties	Type	Description
----- ----- -----		
firstDayOfWeek	number	Sets the first day of the week on recurrence editor.
startDate	Date	Sets the start date from which date the recurrence event starts.
dateFormat	string	Sets the specific date format on recurrence editor.
locale	string	Sets the locale to be applied on recurrence editor.
cssClass	string	Allows styling to be applied on recurrence editor with custom class names.
enableRtl	boolean	Allows recurrence editor to render in RTL mode.
minDate	Date	Sets the minimum date on recurrence editor.
maxDate	Date	Sets the maximum date on recurrence editor.
value	string	Sets the recurrence rule value on recurrence editor.
selectedType	number	Sets the specific repeat type on the recurrence editor.

Customizing the End Type Option in Editor

By default, there are 3 types of end options available in the recurrence editor such as:

- Never
- Until
- Count

It is possible to customize the recurrence editor to display only the specific end options, such as the **Until** and **Count** options alone, by setting the appropriate [endTypes](#) option.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { RecurrenceEditorComponent } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [
    RecurrenceEditorModule,
    DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-recurrenceeditor #recurrenceObj></ejs-recurrenceeditor>`
})
export class AppComponent {
  @ViewChild('recurrenceObj')
```

```

public recurrenceObj?: RecurrenceEditorComponent;
ngAfterViewInit() {
  (this.recurrenceObj as RecurrenceEditorComponent ).selectedType = 1;
  (this.recurrenceObj as RecurrenceEditorComponent | any ).endTypes =
  ['until', 'count'];
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Accessing the recurrence rule string

The recurrence rule is usually generated based on the options selected from the recurrence editor and also it follows the [iCalendar](#) specifications. The generated recurrence rule string is a valid one to be used with the Scheduler event's recurrence rule field.

There is a `change` event available in recurrence editor, that triggers on every time the fields of recurrence editor tends to change. Within this event argument, you can access the generated recurrence value through the `value` option as shown in the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { RecurrenceEditorChangeEventArgs } from '@syncfusion/ej2-angular-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({
  imports: [
    RecurrenceEditorModule,
    DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<div style="padding-bottom:15px;">
    <label id="rule-label">Rule Output</label>
    <div class="rule-output-container">
      <div id="rule-output">{{selectRule}}</div>
    </div>
    <ejs-recurrenceeditor (change)="onChange($event)"></ejs-recurrenceeditor>
  `
})
export class AppComponent {
  public selectRule: string = 'Select Rule';
}

```

```

onChange(args: RecurrenceEditorChangeEventArgs): void {
  if (!isNullOrUndefined(args.value)) {
    if (args.value == "") {
      this.selectRule = 'Select Rule';
    } else {
      this.selectRule = args.value;
    }
  }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set specific value on recurrence editor

It is possible to display the recurrence editor with specific options loaded initially, based on the rule string that we provide. The fields of recurrence editor will change its values accordingly, when we provide a particular rule through the `setRecurrenceRule` method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
import { RecurrenceEditorChangeEventArgs } from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    RecurrenceEditorModule,
    DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<div class="content-wrapper recurrence-editor-wrap">
<div style="padding-bottom:15px;">
  <label>Rule Output</label>
  <div class="rule-output-container">
    <div id="rule-output">{{selectRule}}</div>
  </div>
</div>
<ejs-recurrenceeditor (change)="onChange($event)"
value="FREQ=DAILY;INTERVAL=2;COUNT=8"></ejs-recurrenceeditor>
</div>`
})
export class AppComponent {
  public selectRule: string = 'FREQ=DAILY;INTERVAL=2;COUNT=8';
}

```

```

    public onChange(args: RecurrenceEditorChangeEventArgs): void {
        if (!isNullOrUndefined(args.value)) {
            this.selectRule = args.value;
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Recurrence date generation

You can parse the `recurrenceRule` of an event to generate the date instances on which that particular event is going to occur, using the `getRecurrenceDates` method. It generates the dates based on the `recurrenceRule` that we provide. The parameters to be provided for `getRecurrenceDates` method are as follows.

Field name	Type	Description
----- ----- -----		
<code>startDate</code>	Date	Appointment start date.
<code>rule</code>	String	Recurrence rule present in an event object.
<code>excludeDate</code>	String	Date collection (in ISO format) to be excluded. It is optional .
<code>maximumCount</code>	Number	Number of date count to be generated. It is optional .
<code>viewDate</code>	Date	Current view range's first date. It is optional .

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { RecurrenceEditor, RecurrenceEditorChangeEventArgs } from
 '@syncfusion/ej2-angular-schedule';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({
  imports: [

    RecurrenceEditorModule,
    DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<div style="padding-bottom:15px;">
<label id="rule-label">Date Collections</label>
<div class="rule-output-container">

```

```

        <div id="rule-output">
        <div *ngFor="let date of dateArray">{{date}}</div>
        </div>
    </div>
</div>
<ejs-recurrenceeditor #recurrenceObj [value]="value"
(change)="onChange($event)"></ejs-recurrenceeditor>`
})
export class AppComponent {
    @ViewChild('recurrenceObj') public recObject: RecurrenceEditor;
    public value = 'FREQ=DAILY;INTERVAL=1';
    public dateArray: string[] = [];
    public selectRule: string = 'Select Rule';
    onChange(args: RecurrenceEditorChangeEventArgs): void {
        if (!isNullOrUndefined(args.value)) {
            this.dateArray = [];
            if (args.value == '') {
                this.dateArray.push(this.selectRule.toString());
            } else {
                let dates: number[] = this.recObject.getRecurrenceDates(
                    new Date(),
                    args.value
                );
                for (let i: number = 0; i < dates.length; i++) {
                    this.dateArray.push(new Date(dates[i]).toString());
                }
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Above example will generate two dates January 7, 2018 & January 9 2018 by excluding the in between dates January 8 2018 & January 10 2018, since those dates were given in the exclusion list. Generated dates can then be utilised to create appointments.

Recurrence date generation in server-side

It is also possible to generate recurrence date instances from server-side by manually referring the **RecurrenceHelper** class, which is specifically written and referred from application end to handle this date generation process.

Refer [here](#) for the step by step procedure to achieve date generation in server-side.

Restrict date generation with specific count

In case, if the rule is given in "NEVER ENDS" category, then you can mention the maximum count when you actually want to stop the date generation starting from the provided start date. To do so, provide the appropriate **maximumCount** value within the **getRecurrenceDates** method as shown in the following code example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { RecurrenceEditorModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { NumericTextBoxAllModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit } from '@angular/core';
import { RecurrenceEditor } from '@syncfusion/ej2-angular-schedule';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-inputs';
import { isNullOrUndefined } from '@syncfusion/ej2-base';
@Component({
  imports: [

    RecurrenceEditorModule,
    DropDownListAllModule,
    NumericTextBoxAllModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<div style="padding-bottom:15px;">
<label id="rule-label">Date Collections</label>
<div class="rule-output-container">
  <div id="rule-output">
    <div *ngFor="let date of dateArray">{{date}}</div>
  </div>
</div>
</div>
<ejs-numerictextbox [(value)]='numericValue' (change)=
"onChange($event)"></ejs-numerictextbox>`
})
export class AppComponent implements OnInit {
  public ruleString = 'FREQ=DAILY;INTERVAL=1';
  public numericValue: number = 10;
  public dateArray: string[] = [];
  public recObject: RecurrenceEditor = new RecurrenceEditor();
  ngOnInit(): void {
    let dates: number[] = this.recObject.getRecurrenceDates(
      new Date(2018, 0, 7, 10, 0),
      this.ruleString,
      '20180108T114224Z,20180110T114224Z',
      this.numericValue,
      new Date(2018, 0, 7)
    );
    for (let i: number = 0; i < dates.length; i++) {
      this.dateArray.push(new Date(dates[i]).toString());
    }
  }
  onChange(args: ChangeEventArgs) {
    if (!isNullOrUndefined(args.value)) {
      this.dateArray = [];
      let dates: number[] = this.recObject.getRecurrenceDates(
        new Date(2018, 0, 7, 10, 0),
        this.ruleString,
        '20180108T114224Z,20180110T114224Z',
        args.value,

```

```

        new Date(2018, 0, 7)
    );
    for (let i: number = 0; i < dates.length; i++) {
        this.dateArray.push(new Date(dates[i]).toString());
    }
}
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Localization in Angular Schedule component

The Scheduler integrates different date-time formats and cultures, which allows it to function globally, thus meeting the diverse needs of different regions.

You can adapt the Scheduler to various languages by parsing and formatting the date or number ([Internationalization](#)), adding culture specific customization and translation to the text ([Localization](#)).

Globalization

The Internationalization library provides support for formatting and parsing the number, date, and time by using the official [Unicode CLDR](#) JSON data and also provides the `loadCldr` method to load the culture specific CLDR JSON data.

By default, Scheduler is set to follow the English culture ('en-US'). If you want to go with 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). For more information about CLDR-Data, refer to this [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`.

- Now import the installed CLDR JSON data into the `app.component.ts` file. To import JSON data, you need to install the JSON plugin loader. Here, we have used the SystemJS JSON plugin loader.

```
`sh
```

```
npm install systemjs-plugin-json --save-dev
```

- Once installed, configure the `system.config.js` configuration settings as shown in the following code to map the `systemjs-plugin-json` loader.

```
`typescript
/

    • System configuration for Angular samples
    • Adjust as necessary for your application needs.

*/
(function (global) {
System.config({
paths: {
// paths serve as alias
'npm:': 'node_modules/',
"syncfusion:": "node_modules/@syncfusion/", // syncfusion alias
},
// map tells the System loader where to look for things
map: {
// our app is within the app folder
'app': 'app',
// angular bundles
'@angular/core': 'npm:@angular/core/bundles/core.umd.js',
'@angular/common': 'npm:@angular/common/bundles/common.umd.js',
'@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
'@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
'@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',
'@angular/http': 'npm:@angular/http/bundles/http.umd.js',
'@angular/router': 'npm:@angular/router/bundles/router.umd.js',
'@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
// syncfusion bundles
"@syncfusion/ej2-inputs": "syncfusion:ej2-inputs/dist/ej2-inputs.umd.min.js",
"@syncfusion/ej2-calendars": "syncfusion:ej2-calendars/dist/ej2-calendars.umd.min.js",
```

```

"@syncfusion/ej2-lists": "syncfusion:ej2-lists/dist/ej2-lists.umd.min.js",
"@syncfusion/ej2-base": "syncfusion:ej2-base/dist/ej2-base.umd.min.js",
"@syncfusion/ej2-schedule": "syncfusion:ej2-schedule/dist/ej2-schedule.umd.min.js",
"@syncfusion/ej2-data": "syncfusion:ej2-data/dist/ej2-data.umd.min.js",
"@syncfusion/ej2-buttons": "syncfusion:ej2-buttons/dist/ej2-buttons.umd.min.js",
"@syncfusion/ej2-popups": "syncfusion:ej2-popups/dist/ej2-popups.umd.min.js",
"@syncfusion/ej2-navigations": "syncfusion:ej2-navigations/dist/ej2-navigations.umd.min.js",
"@syncfusion/ej2-dropdowns": "syncfusion:ej2-dropdowns/dist/ej2-dropdowns.umd.min.js",
"@syncfusion/ej2-splitbuttons": "syncfusion:ej2-splitbuttons/dist/ej2-splitbuttons.umd.min.js",
"@syncfusion/ej2-angular-base": "syncfusion:ej2-angular-base/dist/ej2-angular-base.umd.min.js",
"@syncfusion/ej2-angular-schedule": "syncfusion:ej2-angular-schedule/dist/ej2-angular-
schedule.umd.min.js",
"cldr-data": 'npm:cldr-data',
"plugin-json": "npm:systemjs-plugin-json/json.js",
// other libraries
'rxjs':          'npm:rxjs',
'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js'
},
meta: {
  '*.json': { loader: 'plugin-json' }
},
// packages tells the System loader how to load when no filename and/or no extension
packages: {
  app: {
    defaultExtension: 'js',
    meta: {
      './*.js': {
        loader: 'systemjs-angular-loader.js'
      }
    }
  },
  "cldr-data": { main: 'index.js', defaultExtension: 'js' },
  rxjs: {

```

```
defaultExtension: 'js'
}
}
});
})(this);
`
```

- Now import the required cultures from the installed location to `app.component.ts` file as given in the following code example.

```
`typescript
//import 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/fr-CH/ca-gregorian.json';
import * as numbers from 'cldr-data/main/fr-CH/numbers.json';
import * as timeZoneNames from 'cldr-data/main/fr-CH/timeZoneNames.json';
// Angular CLI 8.0 and above versions
loadCldr(numberingSystems['default'], gregorian['default'], numbers['default'],
timeZoneNames['default']);
// Angular CLI 8.0 below versions
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
`
```

- Set the culture to Scheduler by using the `locale` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService, EventSettingsModel } from '@syncfusion/ej2-
angular-schedule'
import { Component } from '@angular/core';
import { scheduleData } from './datasource';
import { loadCldr } from '@syncfusion/ej2-base';
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';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
@Component({
```

```

imports: [
    ScheduleModule
],
providers: [DayService,
            WeekService,
            WorkWeekService,
            MonthService,
            AgendaService,
            MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px' locale='fr-CH'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[views]='views'></ejs-schedule>`
}))
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Localizing the static Scheduler text

[Localization](#) library allows to display all the static text, date content, and time mode of the Scheduler following the localized language. To achieve this, set the `locale` property of Scheduler, as well as define the translation text of static words of Scheduler through the `load` method.

For example, the following code example lets you to define the French translation words for all the static words used in Scheduler.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService,
MonthService, AgendaService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
import { L10n, loadCldr } from '@syncfusion/ej2-base';
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';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);

```

```

L10n.load({
  'fr-CH': {
    'schedule': {
      'day': 'journée',
      'week': 'La semaine',
      'workWeek': 'Semaine de travail',
      'month': 'Mois',
      'today': 'Aujourd`hui'
    },
    'calendar': {
      'today': 'Aujourd`hui'
    }
  }
});
@Component({
  imports: [
    ScheduleModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService,
    AgendaService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px' locale='fr-CH'
[selectedDate]='selectedDate' [eventSettings]='eventSettings'
[views]='views'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

The localized words for static text used in Scheduler and Recurrence Editor can be referred from the following code.

```
`typescript
```

```
L10n.load({
```

```
"en": {
```

```
"schedule": {  
  "day": "Day",  
  "week": "Week",  
  "workWeek": "Work Week",  
  "month": "Month",  
  "year": "Year",  
  "agenda": "Agenda",  
  "weekAgenda": "Week Agenda",  
  "workWeekAgenda": "Work Week Agenda",  
  "monthAgenda": "Month Agenda",  
  "today": "Today",  
  "noEvents": "No events",  
  "emptyContainer": "There are no events scheduled on this day.",  
  "allDay": "All day",  
  "start": "Start",  
  "end": "End",  
  "more": "more",  
  "close": "Close",  
  "cancel": "Cancel",  
  "noTitle": "(No Title)",  
  "delete": "Delete",  
  "deleteEvent": "This Event",  
  "deleteMultipleEvent": "Delete Multiple Events",  
  "selectedItems": "Items selected",  
  "deleteSeries": "Entire Series",  
  "edit": "Edit",  
  "editSeries": "Entire Series",  
  "editEvent": "This Event",  
  "createEvent": "Create",  
  "subject": "Subject",  
  "addTitle": "Add title",  
  "moreDetails": "More Details",  
  "save": "Save",
```



```
"editContent": "How would you like to change the appointment in the series?",
"deleteContent": "Are you sure you want to delete this event?",
"deleteMultipleContent": "Are you sure you want to delete the selected events?",
"newEvent": "New Event",
"title": "Title",
"location": "Location",
"description": "Description",
"timezone": "Timezone",
"startTimezone": "Start Timezone",
"endTimezone": "End Timezone",
"repeat": "Repeat",
"saveButton": "Save",
"cancelButton": "Cancel",
"deleteButton": "Delete",
"recurrence": "Recurrence",
"wrongPattern": "The recurrence pattern is not valid.",
"seriesChangeAlert": "Do you want to cancel the changes made to specific instances of this series and match it to the whole series again?",
"createError": "The duration of the event must be shorter than how frequently it occurs. Shorten the duration, or change the recurrence pattern in the recurrence event editor.",
"sameDayAlert": "Two occurrences of the same event cannot occur on the same day.",
"editRecurrence": "Edit Recurrence",
"repeats": "Repeats",
"alert": "Alert",
"startEndError": "The selected end date occurs before the start date.",
"invalidDateError": "The entered date value is invalid.",
"blockAlert": "Events cannot be scheduled within the blocked time range.",
"ok": "Ok",
"yes": "Yes",
"no": "No",
"occurrence": "Occurrence",
"series": "Series",
"previous": "Previous",
"next": "Next",
```

```
"timelineDay": "Timeline Day",
"timelineWeek": "Timeline Week",
"timelineWorkWeek": "Timeline Work Week",
"timelineMonth": "Timeline Month",
"timelineYear": "Timeline Year",
"editFollowingEvent": "Following Events",
"deleteTitle": "Delete Event",
"editTitle": "Edit Event",
"beginFrom": "Begin From",
"endAt": "End At",
"expandAllDaySection": "Expand-all-day-section",
"collapseAllDaySection": "Collapse-all-day-section"
},
"recurrenceeditor": {
  "none": "None",
  "daily": "Daily",
  "weekly": "Weekly",
  "monthly": "Monthly",
  "month": "Month",
  "yearly": "Yearly",
  "never": "Never",
  "until": "Until",
  "count": "Count",
  "first": "First",
  "second": "Second",
  "third": "Third",
  "fourth": "Fourth",
  "last": "Last",
  "repeat": "Repeat",
  "repeatEvery": "Repeat every",
  "on": "Repeat On",
  "end": "End",
  "onDay": "Day",
```

```

"days": "Day(s)",
"weeks": "Week(s)",
"months": "Month(s)",
"years": "Year(s)",
"every": "every",
"summaryTimes": "time(s)",
"summaryOn": "on",
"summaryUntil": "until",
"summaryRepeat": "Repeats",
"summaryDay": "day(s)",
"summaryWeek": "week(s)",
"summaryMonth": "month(s)",
"summaryYear": "year(s)",
"monthWeek": "Month Week",
"monthPosition": "Month Position",
"monthExpander": "Month Expander",
"yearExpander": "Year Expander",
"repeatInterval": "Repeat Interval"
}
}
});
`

```

Setting date format

Scheduler can be used with all valid date formats and by default it follows the universal date format "MM/dd/yyyy". If the `dateFormat` property is not specified particularly, then it will work based on the locale that is assigned to the Scheduler. As the default locale applied on Scheduler is "en-US", this makes it to follow the "MM/dd/yyyy" pattern.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({

```

```

imports: [
    ScheduleModule,
    ButtonModule
],
providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<ejs-schedule width='100%' height='550px'
[dateFormat]="dateFormat" [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public dateFormat: string = "yyyy/MM/dd";
    public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

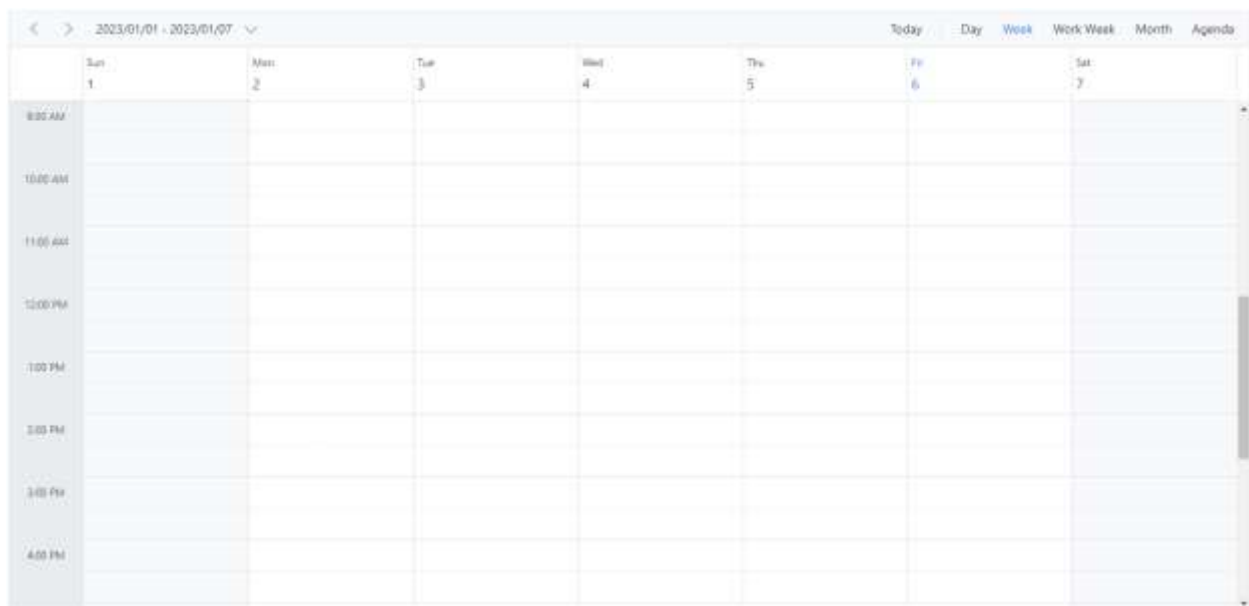
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Setting the time format

Time formats is a way of representing the time value in different string formats in the Scheduler. By default, the time mode of the Scheduler can be either 12 or 24 hours format which is completely based on the **locale** set to the Scheduler. Since the default **locale** value of the Scheduler is en-US, the time mode will be set to 12 hours format automatically. You can also customize the format by using the **timeFormat** property. To know more about the time format standards, refer to the [Date and Time Format](#) section.

Note: **timeFormat** property only accepts the valid time format's.

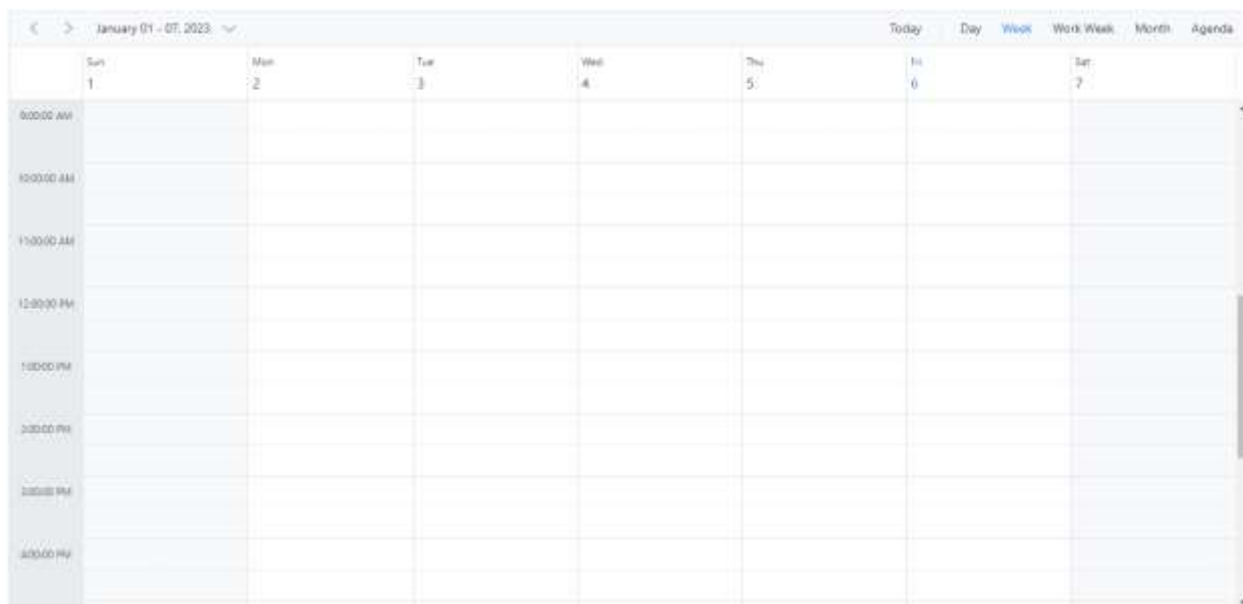
The following example demonstrates the Scheduler component in 24 hours format.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, DayService, WeekService, WorkWeekService, MonthService } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [
    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[timeFormat]="timeFormat" [selectedDate]="selectedDate"
[eventSettings]="eventSettings"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public timeFormat: string = "HH:mm";
  public views: Array<string> = ['Day', 'Week', 'WorkWeek', 'Month'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```



Note: To import the required cultures from the installed location to `app.component.ts` file as given in the following code example.

`typescript

//import 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/fr-CH/ca-gregorian.json';

import * as numbers from 'cldr-data/main/fr-CH/numbers.json';

import * as timeZoneNames from 'cldr-data/main/fr-CH/timeZoneNames.json';

// Angular CLI 8.0 and above versions

loadCldr(numberingSystems['default'], gregorian['default'], numbers['default'],
timeZoneNames['default']);

// Angular CLI 8.0 below versions

loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);

,

Displaying Scheduler in RTL mode

The Scheduler layout and its behavior can be changed as per the common RTL (Right to Left) conventions by setting `enableRtl` to `true`. By doing so, the Scheduler will display its usual layout from right to left. Its default value is `false`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
```

```

import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[enableRtl]='enableRtl'
[selectedDate]='selectedDate' [views]='views'
[eventSettings]='eventSettings'></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public enableRtl: boolean = true;
  public views: Array<string> = ['Day', 'Week', 'WorkWeek'];
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
}

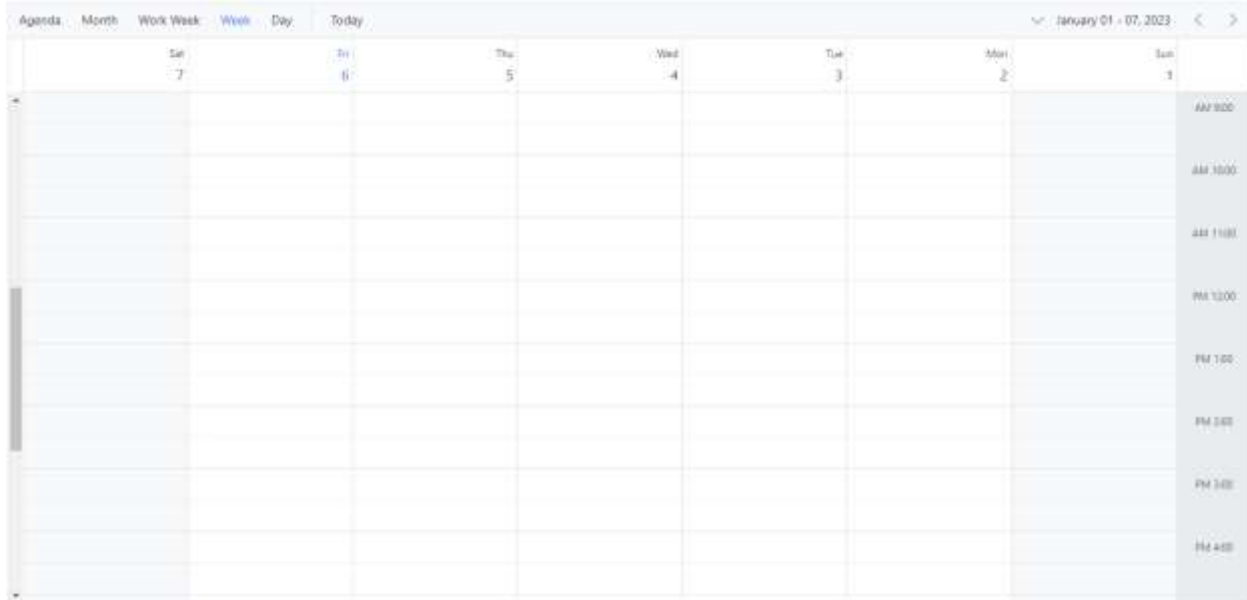
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

See Also

- [How to change first day of the week in the Scheduler](#)

Accessibility in Angular Schedule component

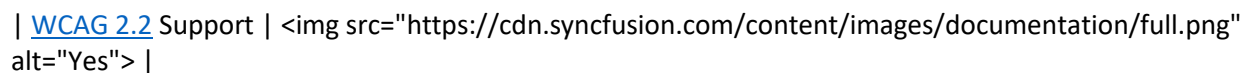
The Scheduler has been designed based on the WAI-ARIA specifications, thus applying the appropriate ARIA roles, states and properties for the Scheduler elements. It is also available with a built-in keyboard navigation support, making it easier for the people who use assistive technologies or who completely rely on the Keyboard support. As per the accessibility standard, the navigated dates, views and other interactive actions performed on the Scheduler will be read out to the target users who use assistive technologies such as screen readers.

The Scheduler makes use of the most required ARIA attributes such as `aria-label` and `role` to support the accessibility in it. To be more accurate, it must be used with an ARIA compliant browser along with the screen reader running from backend.

The accessibility compliance for the Schedule component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes" > |

| [Section 508](#) Support |  alt="Yes" > |

| Screen Reader Support |  alt="Yes" > |


```

| 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 Scheduler parent element is assigned with a role of `main`, to denote it as the main content of a component as well as a unique element of the entire document.

The following ARIA attributes are used in the Scheduler.

Attributes	Description
role="main"	Attribute added to the Scheduler element describes the actual role of the element and denote it as a main and unique content.
role="button"	Attribute is assigned to the appointments of Scheduler, to denote it as a clickable element.
aria-label	Attribute is set to the Scheduler parent element and its default value is Scheduler's current date. On every time, the date is navigated, this attribute is updated with appropriate current date values. It is also assigned to other scheduler UI elements such as previous and next date navigation

buttons depicting its purpose, div element displaying date range in the header bar and appointment elements. |

| aria-labelledby | It indicates editor dialog title to the user through assistive technologies. |

| aria-describedby | It indicates editor dialog content description to the user through assistive technologies. |

| aria-disabled | Attribute is set to the appointment element to indicates the disabled state of the Scheduler.

Keyboard interaction

All the Scheduler actions can be controlled via keyboard keys by using the `allowKeyboardInteraction` property which is set to `true` by default. The following are the standard keys that work on Scheduler.

| Keys | Description |

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

| Alt + j | Focuses the Scheduler element [provided from application end]. |

| Tab | Focuses the first or active item on the Scheduler header bar and then move the focus to the next available event elements. If no events present, then focus moves out of the component. |

| Shift + Tab | Reverse focusing of the Tab key functionality. Inverse focusing of event elements from the last one and then move onto the first or active item on Scheduler header bar and then moves out of the component.

| Enter | Opens the quick info popup on the selected cells or events. |

| Escape | Closes any of the popup that are in open state. |

| Arrow | To move onto the next available cells in either of the needed directions. (left, right, top and right) |

| Shift + Arrow | For multiple cell selection on either direction. |

| Delete | Deletes one or more selected events. |

| Ctrl + Click on events | To select multiple events. |

| Alt + Number (from 1 to 6) | To switch between the views of Scheduler. |

| Ctrl + Left Arrow | To navigate to the previous date period. |

| Ctrl + Right Arrow | To navigate to the next date period. |

| Left or Right Arrow | On pressing any of these keys, when focus is currently on the Scheduler header bar, moves the focus to the previous or next items in the header bar. |

| Space or Enter | It activates any of the focused items. |

| Page Up & Page Down | To scroll through the work cells area. |

| Home | To move the selection to the first cell of Scheduler. |

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Ensuring accessibility

The Scheduler component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Scheduler component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Scheduler component with accessibility tools.

See also

- [Accessibility in Syncfusion components](#)

Scheduler styling in Angular Schedule component

To modify the Scheduler appearance, you need to override the default CSS of Scheduler. Also, you have an option to create your own custom theme using our [Theme Studio](#). Please find the list of CSS classes in Scheduler.

Css class	Purpose
----- -----	
.e-schedule .e-vertical-view .e-work-cells	Work cells in vertical views of scheduler
.e-schedule .e-month-view .e-work-cells	Work cells in month view of scheduler
.e-schedule .e-month-view .e-other-month	Work cells of other month in month view of scheduler
.e-schedule .e-timeline-view .e-work-cells	Work cells in timeline views of scheduler
.e-schedule .e-timeline-month-view .e-work-cells	Work cells in timeline month view of scheduler
.e-schedule .e-timeline-year-view .e-work-cells	Work cells in timeline year view of scheduler
.e-schedule .e-timeline-year-view .e-work-cells.e-other-month	Work cells of other month in timeline year view of scheduler
.e-schedule .e-month-agenda-view .e-work-cells	Work cells in month agenda view of scheduler
.e-schedule .e-month-agenda-view .e-other-month	Work cells of other month in month agenda view of scheduler
.e-schedule .e-year-view .e-calendar-wrapper .e-month-calendar.e-calendar .e-other-month	Work cells of other month in year view of scheduler
.e-schedule .e-vertical-view .e-all-day-cells	All day cells in vertical views of scheduler
.e-schedule .e-vertical-view .e-work-hours	Work hour cells in vertical views of scheduler
.e-schedule .e-month-view .e-work-days	Work day cells in month view of scheduler
.e-schedule .e-month-agenda-view .e-work-days	Work day cells in month agenda view of scheduler
.e-schedule .e-timeline-view .e-work-hours	Work hour cells in timeline views of scheduler

| .e-schedule .e-timeline-month-view .e-work-days | Work day cells in timeline month view of scheduler |

| .e-schedule .e-timeline-year-view .e-work-cells.e-work-days | Work day cells in timeline year view of scheduler |

| .e-schedule .e-vertical-view .e-day-wrapper .e-appointment | Appointment in vertical views of scheduler |

| .e-schedule .e-vertical-view .e-all-day-appointment-wrapper .e-appointment | All day Appointment in vertical views of scheduler |

| .e-schedule .e-month-view .e-appointment | Appointment in month view of scheduler |

| .e-schedule .e-timeline-view .e-appointment | Appointment in timeline views of scheduler |

| .e-schedule .e-timeline-month-view .e-appointment | Appointment in timeline month view of scheduler |

| .e-schedule .e-timeline-year-view .e-event-table .e-appointment | Appointment in timeline year view of scheduler |

| .e-schedule .e-year-view .e-calendar-wrapper .e-month-calendar.e-calendar .e-appointment | Appointment in year view of scheduler |

| .e-schedule .e-agenda-view .e-appointment | Appointment in agenda view of scheduler |

| .e-schedule .e-month-agenda-view .e-appointment-indicator | Appointment in month agenda view of scheduler |

| .e-schedule .e-block-appointment | Block appointment in scheduler |

| .e-schedule .e-read-only | Read only appointment in scheduler. |

| e-appointment-border | Appointment which are currently selected, use the appointment class hierarchical based on your views. |

| e-selected-cells | work cells which are currently selected, use the work cell class hierarchical based on your views. |

| e-header-cells | Header cells of scheduler, use the work cells hierarchical based on your views. |

| .e-schedule .e-vertical-view .e-resource-cells| Resource cells in vertical views of scheduler. |

| .e-schedule .e-month-view .e-resource-cells| Resource cells in month view of scheduler. |

| .e-schedule .e-timeline-view .e-resource-cells | Resource cells in timeline views of scheduler. |

| .e-schedule .e-timeline-month-view .e-resource-cells| Resource cells in timeline month view of scheduler. |

| e-parent-node | Parent resource cells in timeline views of scheduler. |

| e-child-node | Child resource cells in timeline views of scheduler. |

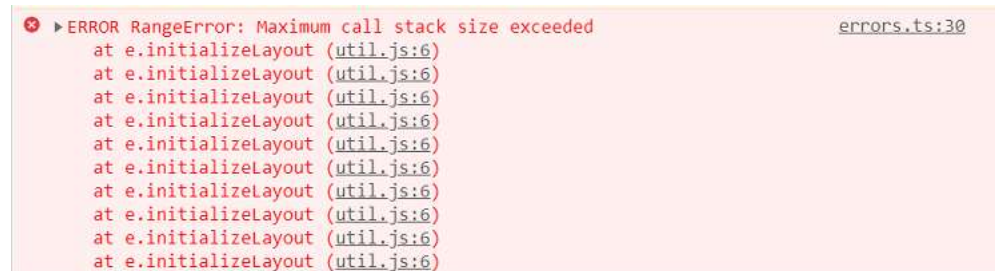
You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

Frequently asked questions in Angular Schedule component

In this article, you can find some frequently asked questions and corresponding solutions while getting hands-on experience with scheduler control.

Maximum call stack size exceeded

Error Image:



Solution:

The above error occurs when using scheduler views that were not imported into the project. You can resolve this issue by importing the required view modules.

In the below code, `Day` option is used without injecting, So, it throws the above error. You can resolve this problem by simply injecting the day module in below code.

`typescript

```
import { Component } from '@angular/core';

import { AgendaService, TimelineViewsService, TimelineMonthService, EventSettingsModel } from
 '@syncfusion/ej2-angular-schedule';

@Component({
  selector: "app-root",
  providers: [ AgendaService, TimelineViewsService, TimelineMonthService],
  // specifies the template string for the Schedule component
  template: `
    <ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate" [views]="views"
    [eventSettings]="eventSettings">
    <e-resources>
    <e-resource field="OwnerId" title="Owner" name="Owners"
    [dataSource]="ownerDataSource" [allowMultiple]="allowMultipleOwner"
    textField="OwnerText" idField="Id" colorField="OwnerColor">
    </e-resource>
    </e-resources>
    </ejs-schedule>`
})
```

```
export class AppComponent {  
  public selectedDate: Date = new Date(2021, 7, 1);  
  // Day view service not imported but still used  
  public views: Array<string> = ['Day', 'TimelineWeek', 'TimelineMonth', 'Agenda'];  
  public eventSettings: EventSettingsModel = {  
    dataSource: resourceData  
  };  
  public allowMultipleOwner: Boolean = true;  
  public ownerDataSource: Object[] = [  
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },  
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },  
    { OwnerText: 'Michael', Id: 3, OwnerColor: '#7499e1' }  
  ];  
}
```

If `ScheduleAllModule` was injected from module files, the above issue is not thrown.

Grouping with empty resources

Grouping without providing any resource data will throw the following problems.

- Normal(vertical) views are rendered, but you are not able to perform CRUD operations
- Timeline views not at all render and shows empty scheduler table

So, we suggest to avoid grouping with empty resources in the scheduler.

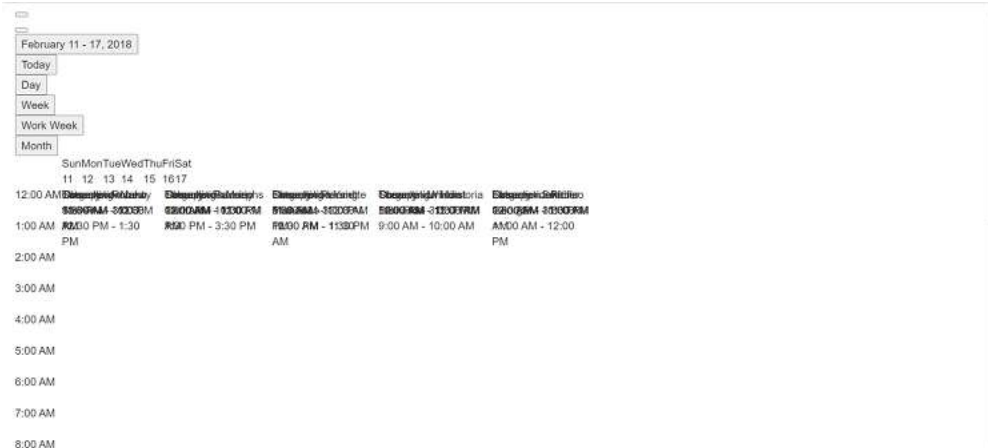
Not providing e-field in editor template

Error: While using editor template, value of `e-field` is missing in editor window.

Solution: `e-field` value is mandatory, we need to add it. Please refer [here](#) for more info.

Missing CSS reference

Error Image:

**Solution:**

The above problem occurs when missing CSS references for the scheduler in a project. You can resolve this issue by providing proper CSS for the scheduler.

```
`html
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>EJ2 Angular Schedule Sample</title>
<!-- scheduler CSS is referred from this link -->
<link href="https://cdn.syncfusion.com/ej2/material.css" rel="stylesheet">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
<app-root></app-root>
</body>
</html>
`
```

QuickInfoTemplate at bottom

When using the `quickInfoTemplate` in scheduler, sometimes quickinfo popup not shown fully at the bottom area of scheduler. You can resolve this by using `cellClick` and `eventClick` events and below code snippet.

```
`typescript
template: "<ejs-schedule #schedule height='650px' (cellClick)='onClick($event)' (eventClick)='onClick($event)'></ejs-schedule>"
```

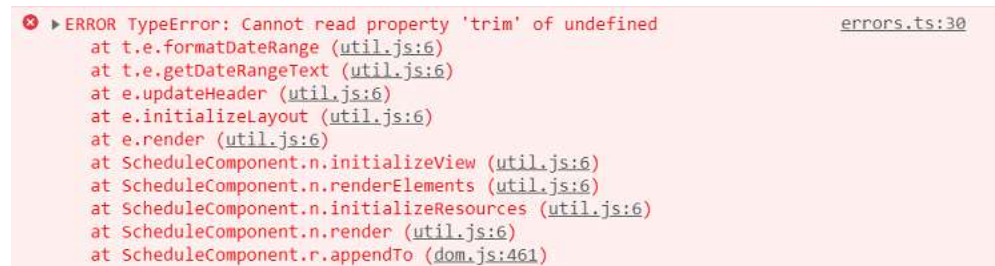
```

.
.
.
public eventAdded: boolean = false;
onClick(args: EventClickArgs): void {
  if (!this.eventAdded) {
    let popupInstance = (document.querySelector(".e-quick-popup-wrapper") as any).ej2_instances[0];
    popupInstance.open = () => {
      popupInstance.refreshPosition();
    };
    this.eventAdded = true;
  }
}

```

Not importing culture files while using localization

Error Image:



While using [locale](#) in scheduler, not importing the required culture files properly throws the problem.

Solution: Properly add and import the culture files (numberingSystems, timeZoneNames, loadCldr, L10n etc.,) in your project will resolve the problem.

`typescript

```

import { loadCldr, L10n } from '@syncfusion/ej2-base';
import * as numberingSystems from './culture-files/numberingSystems.json';
import * as gregorian from './culture-files/ca-gregorian.json';
import * as numbers from './culture-files/numbers.json';
import * as timeZoneNames from './culture-files/timeZoneNames.json';
loadCldr(numberingSystems, gregorian, numbers, timeZoneNames);
L10n.load({
  'en-GB': {

```



```

schedule: {
  day: 'Day',
  week: 'Week',
  workWeek: 'Work Week',
  month: 'Month'
}
});

```

Ej1 api migration in Angular Schedule component

This topic shows the API equivalent of JS2 Scheduler component to be used, while migrating your project that uses JS1 Scheduler.

Scheduler

Properties

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To change the display of days count in agenda view | **Property:** *agendaViewSettings.daysInAgenda*

 <ej-schedule id="Schedule" currentview="currentview"

[agendaViewSettings]="AgendaSettings">
</ej-schedule>
TS
this.AgendaSettings =

{
daysInAgenda: 5}; | **Property:** *agendaDaysCount*

 <ejs-schedule

[(currentView)]="currentView" [agendaDaysCount]= "agendaDaysCount">
</ejs-

schedule>
TS
this.currentView = ['Agenda'];
this.agendaDaysCount=7;|

| Preventing deletion of appointment | **Property:** *allowDelete*

 <ej-schedule id="Schedule"

[allowDelete]="false">
</ej-schedule> | Not applicable |

| Allows dragging and dropping of appointments | **Property:** *allowDragAndDrop*

<ej-

schedule id="Schedule" [allowDragAndDrop]="false">
</ej-schedule>| **Property:**

allowDragAndDrop

 <ejs-schedule [allowDragAndDrop]="false">
</ejs-schedule> |

| Enabling inline editing of appointments | **Property:** *allowInline*

 <ej-schedule

id="Schedule" [allowInline]="false">
</ej-schedule> | Not applicable |

| Allow keyboard interactions | **Property:** *allowKeyboardNavigation*

<ej-schedule

id="Schedule" [allowKeyboardNavigation]="false">
</ej-schedule> | **Property:**

allowKeyboardInteraction

 <ejs-schedule [allowKeyboardInteraction]="false">
</ejs-

schedule> |

| Enable resizing of appointments | **Property:** *enableAppointmentResize*

 <ej-schedule

id="Schedule" [enableAppointmentResize]="false">
</ej-schedule> | **Property:** *allowResizing*

 <ejs-schedule [allowResizing]="false">
</ej-schedule> |

| Blocking time intervals | **Property:** *blockoutSettings*

 <ej-schedule id="Schedule"
[blockoutSettings.enable]="true" [blockoutSettings.dataSource]=blockDataSource
blockoutSettings.id="id" blockoutSettings.startTime="BlockStartTime"
blockoutSettings.endTime="BlockEndTime" blockoutSettings.subject="BlockSubject"
blockoutSettings.isBlockAppointment="IsBlockAppointment">
</ej-
schedule>
TS
this.blockDataSource = [{
BlockId: 101,
BlockStartTime: new
Date(2017, 4, 5, 10, 0),
BlockEndTime: new Date(2017, 4, 5, 11, 0),
BlockSubject:
"Service",
IsBlockAppointment: true}]; | **Property:** *isBlock*

 <ejs-schedule
[eventSettings]="eventSettings">
</ejs-schedule>
TS
public data: Object[] =
[{
Id: 2,
Subject: 'Paris',
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new D
ate(2018, 1, 15, 12, 30),
IsAllDay: false,
IsBlock: true,
});
this.eventSettings=
{
dataSource: this.data}; |

| Categorizing the appointments | **Property:** *categorizeSettings*

 <ej-schedule
id="Schedule" appointmentSettings.categorize="Categorize" [categorizeSettings.enable]="true"
[categorizeSettings.allowMultiple]="true" [categorizeSettings.dataSource]="categorizeData"
categorizeSettings.text="text" categorizeSettings.id="id" categorizeSettings.color="color"
categorizeSettings.fontColor="fontColor">
</ej-schedule>
TS
this.dataSource =
[{
Id: 1,
Subject: "Talk with Nature",
StartTime: new Date(2017, 11, 5, 10,
0),
EndTime: new Date(2017, 11, 5, 11, 0),
Categorize: "1"
});
this.categorizeData
= [{
text: "Blue Category",
id: 1,
color: "#43b496",
fontColor: "#ffffff"
}); | Not
applicable |

| Setting cell height | **Property:** *cellHeight*

 <ej-schedule id="Schedule"
cellHeight="40px">
</ej-schedule> | Not applicable |

| Cell template | **Property:** *workCellsTemplateId*

 <ej-schedule id="Schedule"
allDayCellsTemplateId="#allDayTemplate" workCellsTemplateId="#workTemplate">
</ej-
schedule> | **Property:** *cellTemplate*

 <ejs-schedule width='100%'
height='650px'>
<ng-template #cellTemplate>
You can add template elements here</ng-
template>
</ejs-schedule> |

| Setting cell width | **Property:** *cellWidth*

 <ej-schedule id="Schedule"
cellWidth="100px">
</ej-schedule> | Not applicable |

| CSS class | **Property:** *cssClass*

 <ej-schedule id="Schedule" e-cssClass=
"customStyle">
</ej-schedule> | **Property:** *cssClass*

 <ejs-schedule
cssClass="customStyle">
</ej-schedule> |

| Enabling Context-menu option | **Property:** *contextMenuSettings*

 <ej-schedule
id="Schedule" [contextMenuSettings.enable]="true"
[contextMenuSettings.menuItems]=scheduleMenuItems>
TS
this.scheduleMenuItems =
{
appointment: [{
id: "open",
text: "Open Appointment"}]
}; | Not applicable |

| Current view | **Property:** *currentView*

 <ej-schedule id="Schedule"
currentView="workWeek">
</ej-schedule> | **Property:** *currentView*

 <ejs-schedule
[(currentView)]= "currentView">
</ej-schedule>
TS
this.currentView='Week'; |

| Date format | **Property:** *dateFormat*

 <ej-schedule id="Schedule" dateFormat="yyyy/MM/dd">
</ej-schedule> | **Property:** *dateFormat*

 <ejs-schedule [dateFormat]="dateFormat"></ejs-schedule>
TS
this.dateFormat= "yyyy/MM/dd"; |

| Date header template | **Property:** *dateHeaderTemplateId*

 <ej-schedule id="Schedule" dateHeaderTemplateId="#dateTemplate">
</ej-schedule> | <ejs-schedule width='100%' height='650px'>
<ng-template #dateHeaderTemplate>
You can add template elements here
</ng-template>
</ejs-schedule> |

| Editor template | Not Applicable | **Property:** *editorTemplate*

 <ejs-schedule width='100%' height='650px'>
<ng-template #editorTemplate>
You can add template elements here
</ng-template>
</ejs-schedule> |

Enable load on demand | **Property:** *enableLoadOnDemand*

 <ej-schedule id="Schedule" [enableLoadOnDemand]="false">
</ej-schedule> | Not applicable |

| Enable persistence | **Property:** *enablePersistence*

 <ej-schedule id="Schedule" [enablePersistence]="false">
</ej-schedule> | **Property:** *enablePersistence*

 <ejs-schedule [enablePersistence]="false">
</ejs-schedule> |

| Enable RTL | **Property:** *enableRTL*

 <ej-schedule id="Schedule" [enableRTL]="false">
</ej-schedule> | **Property:** *enableRTL*

 <ejs-schedule [enableRTL]="false">
</ejs-schedule> |

| Setting end hour of the scheduler | **Property:** *endHour*

 <ej-schedule id="Schedule" [endHour]="18">
</ej-schedule> | **Property:** *endHour*

 <ejs-schedule [endHour]="endHour">
</ejs-schedule>
TS
this.endHour='18:00'; |

| Setting first day of the week | **Property:** *firstDayOfWeek*

 <ej-schedule id="Schedule" firstDayOfWeek="Tuesday">
</ej-schedule> | **Property:** *firstDayOfWeek*

 <ejs-schedule [firstDayOfWeek]="firstDayOfWeek">
</ejs-schedule>
TS
this.firstDayOfWeek= 1; |

| Height of the scheduler | **Property:** *height*

 <ej-schedule id="Schedule" height="500px">
</ej-schedule> | **Property:** *height*

 <ejs-schedule height='650px'>
</ejs-schedule> |

| Locale | **Property:** *locale*

 <ej-schedule id="Schedule" locale="fr-CH">
</ej-schedule> | **Property:** *locale*

 <ejs-schedule locale='fr-CH'>
</ejs-schedule> |

| Priority settings for appointments | **Property:** *prioritySettings*

 <ej-schedule id="Schedule" appointmentSettings.priority="Priority" [prioritySettings.enable]="true" [prioritySettings.dataSource]="priorityData" prioritySettings.text="text" prioritySettings.value="value">
</ej-schedule>
TS
this.priorityData = [{
text: "Low",
value: "low"}]; | Not applicable |

| Read only | **Property:** *readOnly*

 <ej-schedule id="Schedule" [readOnly]="true">
</ej-schedule> | **Property:** *readonly*

 <ejs-schedule [readonly]="readonly">
</ejs-schedule>
TS
this.readonly= true; |

| Reminder settings | **Property:** *reminderSettings*

 <ej-schedule id="Schedule"
[reminderSettings.enable]="true" [reminderSettings.alertBefore]="10">
</ej-schedule> |
Not applicable |

| Resource header template | **Property:** *resourceHeaderTemplateId*

 <ej-schedule
id="Schedule" resourceHeaderTemplateId="#resTemplate"
appointmentSettings.resourceFields="RoomId">
<e-resources>
<e-resource
field="RoomId" title="Room" name="Rooms" [resourceSettings]=resourceData</e-
resource>
</e-resources>
</ej-schedule>
TS
this.resourceData =
{
dataSource: [{
ResourceText: "ROOM1",
id: 1,
ResourceColor:
"orange"}],
text: "ResourceText",
id: "id",
color: "ResourceColor"}; | **Property:**
resourceHeaderTemplate

 <ejs-schedule width='100%' height='650px'>
<ng-
template #resourceHeaderTemplate>
 <e-resources>
<e-resource field='ProjectId'
title='Project' name='Project' [allowMultiple]='allowMultiple' [dataSource]='ProjectDataSource'
textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-
schedule>
TS
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT 1', id: 1, color:
'#cb6bb2'}]; |

| Current date of the scheduler | **Property:** *currentDate*

 <ej-schedule id="Schedule"
[currentDate]=currentDate>
</ej-schedule>
TS
this.currentDate = new Date(2017, 1,
7); | **Property:** *selectedDate*

 <ejs-schedule [selectedDate]="selectedDate">
</ejs-
schedule>
TS
this.selectedDate = new Date(2018, 3, 4); |

| Show all day row | **Property:** *showAllDayRow*

 <ej-schedule id="Schedule"
[showAllDayRow]="true">
</ej-schedule> | Not applicable |

| Show appointment navigator | **Property:** *showAppointmentNavigator*

 <ej-schedule
id="Schedule" [showAppointmentNavigator]="true">
</ej-schedule> | Not applicable |

| Show delete confirmation dialog | **Property:** *showDeleteConfirmationDialog*

 <ej-schedule
id="Schedule" [showDeleteConfirmationDialog]="true">
</ej-schedule> | Not applicable |

| Show header bar | **Property:** *showHeaderBar*

 <ej-schedule
id="Schedule" [showHeaderBar]="false">
</ej-schedule> | **Property:** *showHeaderBar*

 <ejs-schedule [showHeaderBar]="showHeaderBar">
</ejs-
schedule>
TS
this.showHeaderBar=true |

| Show location field in event window | **Property:** *showLocationField*

 <ej-schedule
id="Schedule" [showLocationField]="false">
</ej-schedule> | Not applicable |

| Show time zone fields in event window | **Property:** *showTimeZoneFields*

 <ej-schedule
id="Schedule" [showTimeZoneFields]="false">
</ej-schedule> | Not applicable |

| Show previous and next month dates in month view | **Property:** *showNextPrevMonth*

 <ej-
schedule id="Schedule" [showNextPrevMonth]="false">
</ej-schedule> | Not applicable |

| Show overflow button | **Property:** *showOverflowButton*

 <ej-schedule id="Schedule"
[showOverflowButton]="true">
</ej-schedule> | **Property:** *rowAutoHeight*

 <ejs-
schedule [rowAutoHeight]="true">
</ej-schedule> |

| Show quick popup | **Property:** *showQuickWindow*

 <ej-schedule id="Schedule" [showQuickWindow]="false">
</ej-schedule> | **Property:** *showQuickInfo*

 <ejs-schedule [showQuickInfo]="false">
</ejs-schedule> |

| Show current time indicator | **Property:** *showCurrentTimeIndicator*

 <ej-schedule id="Schedule" [showCurrentTimeIndicator]="false">
</ej-schedule> | **Property:** *showTimeIndicator*

 <ejs-schedule [showTimeIndicator]="showTimeIndicator">
</ejs-schedule>
TS
this.showTimeIndicator=false;|

| Show week number | Not Applicable | **Property:** *showWeekNumber*

 <ejs-schedule [showWeekNumber]="showWeekNumber">
</ejs-schedule>
TS
this.showWeekNumber=true;|

| Show weekend days | **Property:** *showWeekend*

 <ej-schedule id="Schedule" [showWeekend]="false">
</ej-schedule> | **Property:** *showWeekend*

 <ejs-schedule [showWeekend]="showWeekend">
</ejs-schedule>
TS
this.showWeekend=false;|

| Setting start hour of the scheduler | **Property:** *startHour*

 <ej-schedule id="Schedule" [startHour]="7">
</ej-schedule> | **Property:** *startHour*

 <ejs-schedule [startHour]="startHour">
</ejs-schedule>
TS
this.startHour= '07:00'; |

| Setting time mode on scheduler | **Property:** *timeMode*

 <ej-schedule id="Schedule" timeMode="ej.Schedule.TimeMode.Hour24">
</ej-schedule> | Not applicable |

| Setting timezone for scheduler | **Property:** *timeZone*

 <ej-schedule id="Schedule" timeZone= "UTC +05:30">
</ej-schedule> | **Property:** *timezone*

 <ejs-schedule timezone="UTC">
</ejs-schedule> |

| Views in scheduler | **Property:** *views*

 <ej-schedule id="Schedule" [views]="views">
</ej-schedule>
TS
this.views = ["Day", "Week", "WorkWeek", "Month"]; | **Property:** *views*

 <ejs-schedule width='100%' height='650px'>
<e-views>
<e-view option="Day"></e-view>
<e-view option="Week"></e-view>
<e-view option="WorkWeek"></e-view>
<e-view option="Month"></e-view>
</e-views>
</ejs-schedule> |

| Width of the scheduler | **Property:** *width*

 <ej-schedule id="Schedule" width="70%">
</ej-schedule> | **Property:** *width*

 <ejs-schedule width='100%'>
</ejs-schedule> |

| Working days | **Property:** *workWeek*

 <ej-schedule id="Schedule" [workWeek]="workWeek">
</ej-schedule>
TS
this.workWeek=["Monday", "Tuesday", "Friday"]; | **Property:** *workDays*

 <ejs-schedule [workDays]="workDays">
</ejs-schedule>
TS
this.workDays= [1, 3, 5]; |

| Working hours | **Property:** *workHours*

 <ej-schedule id="Schedule" [workHours.highlight]="true" [workHours.start]="8" [workHours.end]="16">
</ej-schedule> | **Property:** *workHours*

 <ejs-schedule [workHours]="workHours">
</ejs-schedule>
TS
this.workHours= { highlight="true" start: '9:00', end: '11:00' }; |

Resources

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To define resource datasource | **Property:** *resources*

 <ej-schedule id="Schedule" appointmentSettings.resourceFields="RoomId">
<e-resources>
<e-resource field="RoomId" title="Room" name="Rooms" [allowMultiple]="true" [resourceSettings]=ownerData></e-resource>
</e-resources>
</ej-schedule>
TS
this.ownerData = {
dataSource: [{
OwnerText: "Nancy",
id: 1,
OwnerColor: "#f8a398"}],
text: "OwnerText",
id: "id",
color: "OwnerColor"}; |

Property: *resources*

 <ejs-schedule width='100%' height='550px'>
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='resourceDataSource' name='Project' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
TS
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT 1', id: 1, color: '#cb6bb2'}]; |

| Allowing multiple selection of resources in event window | **Property:** *e-allowMultiple*

 <ej-schedule id="Schedule" appointmentSettings.resourceFields="RoomId">
<e-resources>
<e-resource field="RoomId" title="Room" name="Rooms" [allowMultiple]="true" [resourceSettings]=roomData></e-resource>
<e-resource field="OwnerId" title="Owner" name="Owners" [allowMultiple]="true" [resourceSettings]=ownerData></e-resource>
</e-resources>
</ej-schedule>
TS
this.roomData = {
dataSource: [{
RoomText: "Room1",
id: 1,
RoomColor: "#f8a398"},
{ RoomText: "Room2",
id: 2,
RoomColor: "#f8a398"}],
text: "RoomText",
id: "id",
color: "RoomColor"};
this.ownerData = {
dataSource: [{
OwnerText: "Nancy",
id: 1,
OwnerColor: "#f8a398"},
{ OwnerText: "Steven",
id: 2,
OwnerColor: "#f8a398"}],
text: "OwnerText",
id: "id",
color: "OwnerColor"}; | **Property:** *allowMultiple*

 <ejs-schedule width='100%' height='550px'>
<e-resources>
<e-resource field='ProjectId' title='Room' [dataSource]='ProjectDataSource' name='Project' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
TS
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT 1', id: 1, color: '#cb6bb2'},
{ text: 'PROJECT 2', id: 2, color: '#cb6bb2'}]; |

| Setting different work hours for each resource |

 <ej-schedule id="Schedule" appointmentSettings.resourceFields="RoomId" [group]="group">
<e-resources>
<e-resource field="RoomId" title="Room" name="Rooms" [allowMultiple]="true" [resourceSettings]=roomData></e-resource>
</e-resources>
</ej-schedule>
TS
this.group = {
resources: ["Rooms"]};
this.roomData = {
dataSource: [{ text: "Nancy", id: 1, color: "#ffaa00", on: 10, off: 18, customDays: ["monday", "wednesday", "friday"]
}],
text: "text", id: "id", groupId: "groupId", color: "color", start: "on", end: "off", workWeek: "customDays"}; | <ejs-schedule width='100%' height='550px' [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='ProjectDataSource' name='Project' textField='text' idField='id' colorField='color' workDaysField= 'workDays' startHourField= 'startHour' endHourField=

```
'endHour'><br></e-resource><br><e-resource field='CategoryId' title='Category'
[dataSource]='CategoryDataSource' [allowMultiple]='allowMultiple' name='Categories'
textField='text' idField='id' colorField='color' workDaysField='workDays' startHourField='
startHour' endHourField='endHour'><br></e-resource><br></e-resources><br></ejs-
schedule><br>TS<br>this.group = {resources: ['Project']};<br>this.ProjectDataSource: Object[] =
[<br>{ text: 'PROJECT 1', id: 1, color: '#cb6bb2',workDays: [1, 2, 4, 5],startHour: '07:00',
endHour: '13:00'}];<br>this.CategoryDataSource: Object[] = [<br>{ text: 'Development', id: 1,
color: '#1aaa55',workDays: [1, 2, 3, 4, 5],startHour: '09:00', endHour: '13:00'}];<br>]; |
```

Group

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To group the resources in scheduler layout | **Property:** *group*

 <ej-schedule id="Schedule" width="100%" height="525px" appointmentSettings.resourceFields="OwnerId,RoomId" [group]="group">
<e-resources>
<e-resource field="OwnerId" title="Owner" name="Owners" [resourceSettings]="ownerData"></e-resource>
<e-resource field="RoomId" title="Room" name="Rooms" [allowMultiple]="true" [resourceSettings]="roomData"></e-resource>
</e-resources>
</ej-schedule>
TS
this.group = {
resources: ["Owners", "Rooms"]};
this.ownerData = {
dataSource: [{
OwnerText: "Nancy",
id: 1,
OwnerColor: "#f8a398"}],
text: "OwnerText",
id: "id",
color: "OwnerColor"};
this.roomData = {
dataSource: [{
text: "Room1",
id: 1,
color: "#f8a398"}],
text: "text",
id: "id",
color: "color"}; | **Property:** *group*

 <ej-schedule width='100%' height='550px' [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='ProjectDataSource' name='Projects' textField='text' idField='id' colorField='color'>
</e-resource>
<e-resource field='CategoryId' title='Category' [dataSource]='CategoryDataSource' name='Categories' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ej-schedule>
TS
this.group = {resources: ['Project','Category'] };
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT1', id: 1, color: '#cb6bb2'}];
this.CategoryDataSource: Object[] = [
{ text: 'Testing', id: 1, color: '#cb6bb2'}]; |

| Allow group editing | **Property:** *allowGroupEditing*

 <ej-schedule id="Schedule" appointmentSettings.resourceFields="RoomId">
<e-resources>
<e-resource field="RoomId" title="Room" name="Rooms" [allowMultiple]="true" [resourceSettings]="ownerData" [group]="group"></e-resource>
</e-resources>
</ej-schedule>
TS
this.group = {
resources: ["Owners", "Rooms"], [allowGroupEditing]="true"};
this.ownerData = {
dataSource: [{
OwnerText: "Nancy",
id: 1,
OwnerColor: "#f8a398"}],
{ OwnerText: "Steven",
id: 2,
OwnerColor: "#f8a398"}],
text: "OwnerText",
id: "id",
color: "OwnerColor"}; | **Property:** *allowGroupEdit*

 <ej-schedule width='100%' height='550px' [group]="group">
<e-resources>
<e-resource field='RoomId' title='Room' [dataSource]='resourceDataSource' name='Room' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ej-schedule>
TS
this.group = {allowGroupEdit: true,

```
resources:['Room'] };<br>this.ProjectDataSource: Object[] = [<br>{ text: 'PROJECT1', id: 1, color: '#cb6bb2'},<br>{ text: 'PROJECT2', id: 2, color: '#cb6bb2'},<br>{ text: 'PROJECT3', id: 2, color: '#cb6bb2'}]; |
```

| Grouping resources by date | Not applicable | **Property:** *byDate*

 <ejs-schedule [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='ProjectDataSource' name='Project' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
this.group = {byDate: true, resources:['Project'] };
TS
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT1', id: 1, color: '#cb6bb2'},
{ text: 'PROJECT2', id: 2, color: '#cb6bb2'}]; |

| Grouping resources based on its group ID | Not applicable | **Property:** *byGroupId*

 <ejs-schedule width='100%' height='550px' [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Choose Project' name='Projects' [dataSource]='projectDataSource' textField='text' idField='id' colorField='color'>
</e-resource>
<e-resource field='CategoryId' title='Category' name='Categories' [dataSource]='categoryDataSource' [allowMultiple]='allowMultiple' textField='text' idField='id' groupIdField='groupId' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
TS
this.group = {byGroupId: false, resources:['Project','Category'] };
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT1', id: 1, color: '#cb6bb2'},
{ text: 'PROJECT2', id: 2, color: '#cb6bb2'}];
this.categoryDataSource: Object[] = [
{ text: 'Testing', id: 1, groupId: 1, color: '#cb6bb2'},
{ text: 'Development', id: 2, groupId: 2, color: '#cb6bb2'},
{ text: 'Documentation', id: 3, groupId: 2, color: '#cb6bb2'}]; |

| Enabling compact view on mobile mode | Not applicable | **Property:** *enableCompactView*

 <ejs-schedule width='100%' height='550px' [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='ProjectDataSource' name='Project' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
TS
this.group = {enableCompactView: false, resources:['Project'] };
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT1', id: 1, color: '#cb6bb2'}]; |

| Header tooltip template | Not applicable | **Property:** *headerTooltipTemplate*

 <ejs-schedule [group]="group">
<e-resources>
<e-resource field='ProjectId' title='Project' [dataSource]='ProjectDataSource' name='Project' textField='text' idField='id' colorField='color'>
</e-resource>
</e-resources>
</ejs-schedule>
TS
this.group = {headerTooltipTemplate: '#resourceHeader', resources:['Project'] };
this.ProjectDataSource: Object[] = [
{ text: 'PROJECT1', id: 1, color: '#cb6bb2'}]; |

Header Rows

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Adding custom rows in the header in timeline views | Not applicable | **Property:** *HeaderRows*

 <ejs-schedule width='100%' height='650px' [(currentView)]="currentView">
<e-header-rows>
<e-header-row option='Month'>
</e-header-row>
<e-header-row option='Week'>
</e-header-row>
<e-header-row option='Date'>
</e-header-


```
row><br></e-header-rows><br><e-views><br><e-view displayName='Timeline Month'
option='TimelineMonth' [interval]="monthInterval"></e-view><br></e-views><br></ej-
schedule><br>TS<br>this.monthInterval= 12;<br>this.currentView='TimelineMonth'; |
```

TimeScale

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Enabling time scale | **Property:** *timeScale.enable*

 <ej-schedule id="Schedule" [timeScale.enable]="true">
</ej-schedule> | **Property:** *enable*

 <ejs-schedule [timeScale]="timeScale">
</ejs-schedule>
TS
this.timeScale= { enable: true}; |

| Setting major interval on time scale | **Property:** *timeScale.majorSlot*

 <ej-schedule id="Schedule" [timeScale.enable]="true" [timeScale.majorSlot]="60">
</ej-schedule> | **Property:** *interval*

 <ejs-schedule [timeScale]="timeScale">
</ejs-schedule>
TS
this.timeScale= { enable: true, interval: 60}; |

| Setting slot count on time scale | **Property:** *timeScale.minorSlotCount*

 <ej-schedule id="Schedule" [timeScale.enable]="true" [timeScale.majorSlot]="60" [timeScale.minorSlotCount]="6">
</ej-schedule> | **Property:** *slotCount*

 <ejs-schedule [timeScale]="timeScale">
</ejs-schedule>
TS
this.timeScale= { enable: true, interval: 60, slotCount: 6 }; |

| Defining major slot template | **Property:** *timeScale.majorSlotTemplateId*

 <ej-schedule id="Schedule" [timeScale.enable]="true" [timeScale.majorSlot]="60" [timeScale.minorSlotCount]="6" [timeScale.majorSlotTemplateId]="#majorTemplate">
</ej-schedule> | **Property:** *majorSlotTemplate*

 <ejs-schedule [timeScale]="timeScale">
<ng-template #majorSlotTemplate>
You can add template elements here
</ng-template>
</ejs-schedule> |

| Defining minor slot template | **Property:** *timeScale.minorSlotTemplateId*

 <ej-schedule id="Schedule" [timeScale.enable]="true" [timeScale.majorSlot]="60" [timeScale.minorSlotCount]="6" [timeScale.minorSlotTemplateId]="#minorTemplate">
</ej-schedule> | **Property:** *minorSlotTemplate*

 <ejs-schedule [timeScale]="timeScale">
<ng-template #minorSlotTemplate>
You can add template elements here
</ng-template>
</ejs-schedule> |

Quick info templates

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Template for quick popup | Not applicable | **Property:** *quickInfoTemplates*

 <ejs-schedule width='100%' height='550px'>
<ng-template #quickInfoTemplatesHeader>You can add template elements here</ng-template>
<ng-template #quickInfoTemplatesContent>You can add template elements here</ng-template>
<ng-template #quickInfoTemplatesFooter>You can add template elements here</ng-template>
</ejs-schedule> |

Event settings

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Datasource for events | **Property:** *appointmentSettings.dataSource*

 <ej-schedule id="Schedule" [appointmentSettings.dataSource]=dataSource>
</ej-schedule>
TS
this.dataSource = [{
Id: 1,
Subject: "Testing",
StartTime: new Date(2017, 1, 7, 6, 0),
EndTime: new Date(2017, 1, 7, 7, 30)}]; | **Property:** *dataSource*

 <ejs-schedule [eventSettings]="eventSettings">
</ejs-schedule>
TS
public data: Object[] = [{
Id: 2,
Subject: 'Paris',
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new Date(2018, 1, 15, 12, 30),
IsAllDay: false,
RecurrenceID: 10,
});
this.eventSettings={
dataSource: this.data}; |

| Appointment fields | <ej-schedule id="Schedule" [appointmentSettings.dataSource]="appointments">
</ej-schedule>
TS
this.appointments = [{
Id: 1,
Subject: "Testing",
StartTime: new Date(2017, 1, 7, 6, 0),
EndTime: new Date(2017, 1, 7, 7, 30)}]; | <ejs-schedule [eventSettings]="eventSettings">
</ejs-schedule>
TS
public data: Object[] = [{
Id: 2,
Subject: 'Paris',
StartTime: new Date(2018, 1, 15, 10, 0),
EndTime: new Date(2018, 1, 15, 12, 30),
IsAllDay: false,
RecurrenceID: 10,
});
this.eventSettings={
dataSource: this.data,fields: {
id: 'Id',
subject: { name: 'Subject' },
isAllDay: { name: 'IsAllDay' },
location: { name: 'Location' },
description: { name: 'Description' },
recurrenceID : { name: 'RecurrenceID' }
}}; |

| Enabling tooltip for appointments | **Property:** *tooltipSettings.enable*

 <ej-schedule id="Schedule" [tooltipSettings.enable]="true">
</ej-schedule>
Note: Here tooltip setting for events is maintained separately | **Property:** *enableTooltip*

 <ejs-schedule [eventSettings]="eventSettings">
</ejs-schedule>
TS
this.eventSettings={enableTooltip: true}; |

| Tooltip template for appointments | **Property:** *tooltipSettings.templateId*

 <ej-schedule id="Schedule" [tooltipSettings.enable]="true" tooltipSettings.templateId="#tooltipTemplate">
</ej-schedule>
Note: Here tooltip setting for events is maintained separately | **Property:** *tooltipTemplate*

 <ejs-schedule [eventSettings]="eventSettings">
</ejs-schedule>
TS
this.eventSettings={enableTooltip: true, tooltipTemplate: this.tooltipTemplate}; |

| Template for appointments | **Property:** *appointmentTemplateId*

 <ej-schedule id="Schedule" appointmentTemplateId="#appTemplate">
</ej-schedule> | **Property:** *template*

 <ejs-schedule [eventSettings]="eventSettings">
</ej-schedule>
TS
this.eventSettings= {template: this.eventTemplate }; |

| Query | **Property:** *appointmentSettings.query*

 <ej-schedule id="Schedule" [appointmentSettings.dataSource]=dataManager [appointmentSettings.query]=query>
</ej-schedule>
TS
this.dataManager = ej.DataManager({
url: "http://mvc.syncfusion.com/OdataServices/Northwnd.svc/"
});
this.query = ej.Query().from("Events").take(10); | **Property:** *query*

 <ejs-schedule

```
[eventSettings]="eventSettings"><br></ejs-schedule><br>private dataManger: DataManager =
new DataManager({<br>url: 'https://ej2services.syncfusion.com/production/web-
services/api/Schedule',<br>adaptor: new ODataAdaptor });<br>this.eventSettings = {
dataSource: this.dataManger, query: new Query().addParams('ej2schedule', 'true') }; |
```

| Define which resource level color applied to events | Not applicable | **Property:** *resourceColorField*

```
<br><br> <ejs-schedule [eventSettings]="eventSettings" [group]="group"><br><e-
resources><br><e-resource field='RoomId' title='Room' [dataSource]='roomData'
name='Rooms' textField='RoomText' idField='Id' groupIDField='RoomGroupId'
colorField='RoomColor'><br></e-resource><br><e-resource field='OwnerId' title='Owner'
[dataSource]='ownerData' [allowMultiple]='allowMultiple' name='Owners'
textField='OwnerText' idField='Id' groupIDField='OwnerGroupId'
colorField='OwnerColor'><br></e-resource><br></e-resources><br></ejs-schedule><br>public
roomData: Object[] = [<br>{ RoomText: 'ROOM 1', Id: 1, RoomGroupId: 1, RoomColor:
'#cb6bb2'}<br>];<br>public ownerData: Object[] = [<br>{ OwnerText: 'Nancy', Id: 1,
OwnerGroupId: 1, OwnerColor: '#ffaa00'}];<br>this.group = { resources: ['Rooms', 'Owners']
};<br>this.allowMultiple = true;<br>this.eventSettings= {resourceColorField: 'Owners' }; |
```

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To add appointments manually | **Method:** *saveAppointment()*

 export class AppComponent {
@ViewChild('schedule') Schedule: EJComponents<any, any>;
ngAfterViewInit(){
var data = {
Id: 1,
Subject: "Testing",
StartTime: new Date(2014, 4, 5, 10, 00),
EndTime: new Date(2014, 4, 5, 12, 00)};
this.Schedule.widget.saveAppointment(data);
}
} | **Method:** *addEvent()*

```
<br><br> @ViewChild('schedule') Schedule: ScheduleComponent;<br>public data = {<br>Id: 1,<br>Subject: "New Event",<br>StartTime: new Date(2014, 4, 5, 10, 00),<br>EndTime: new Date(2014, 4, 5, 12, 00)};<br>this.Schedule.addEvent(data); |
```

| To add resources dynamically | **Method:** *addResource()*

 export class AppComponent {
@ViewChild('schedule') Schedule: EJComponents<any, any>;
ngAfterViewInit(){
var data = { text: "Paul", id: 1, groupId: 3, color: "#cc99ff" };
var index = 0;
this.Schedule.widget.addResource(data, "Owners", index);
}
} | **Method:** *addResource()*

```
<br><br> @ViewChild('schedule') Schedule: ScheduleComponent;<br>public data = { text: "Paul", id: 3, groupId: 1, color: "#cc99ff" };<br>public index = 0;<br>this.Schedule.addResource(data, "Owners", index); |
```

| databind | Not applicable | **Method:** *dataBind()*

 @ViewChild('schedule') Schedule: ScheduleComponent;
this.Schedule.dataBind() |

| Delete appointment manually | **Method:** *deleteAppointment()*

 export class AppComponent {
@ViewChild('schedule') Schedule: EJComponents<any, any>;
ngAfterViewInit(){
var data = {
Id: 1,
Subject: "Testing",
StartTime: new Date(2014, 4, 5, 10, 00),
EndTime: new Date(2014, 4, 5, 12, 00)};
this.Schedule.widget.deleteAppointment(data);
}
} | **Method:** *deleteEvent()*

```

</></> @ViewChild('schedule') Schedule: ScheduleComponent;<br>public data = {<br> Id:
1,<br>Subject: "Old Event",<br>StartTime: new Date(2014, 4, 5, 10, 00),<br>EndTime: new
Date(2014, 4, 5, 12, 00)};<br>this.Schedule.deleteEvent(data); |

| destroy scheduler | Method: destroy() <br><br> export class AppComponent
{<br>@ViewChild('schedule') Schedule: EJComponents<any, any>;<br>
ngAfterViewInit(){<br>this.Schedule.widget.destroy();<br>}<br>} | Method: destroy() <br><br>
@ViewChild('schedule') Schedule: ScheduleComponent;<br>this.Schedule.destroy() |

| Get cell details | Method: getSlotByElement() <br><br> export class AppComponent
{<br>@ViewChild('schedule') Schedule: EJComponents<any, any>;<br> ngAfterViewInit(){<br>var
$td = $(".e-draggableworkarea table tr td").first();<br>var slotDetails
=this.Schedule.widget.getSlotByElement($td);<br>}<br>} | Method: getCellDetails() <br><br>
@ViewChild('schedule') Schedule: ScheduleComponent;<br> public td =
document.querySelector(".e-work-cells"); <br>public cellDetail = this.Schedule.getCellDetails(td);
|

| Get current view appointments | Method: getCurrentViewAppointments() <br><br> export class
AppComponent {<br>@ViewChild('schedule') Schedule: EJComponents<any, any>;<br>
ngAfterViewInit(){<br> var currApp= this.Schedule.widget.getCurrentViewAppointments(); |
Method: getCurrentViewEvents() <br><br> @ViewChild('schedule') Schedule:
ScheduleComponent;<br>public currApp = this.Schedule.getCurrentViewEvents(); |

| Get entire appointment collection | Method: getAppointments() <br><br> export class
AppComponent {<br>@ViewChild('schedule') Schedule: EJComponents<any, any>;<br>
ngAfterViewInit(){<br> var AppDetails= this.Schedule.widget.getAppointments(); | Method:
getEvents() <br><br> @ViewChild('schedule') Schedule: ScheduleComponent;<br>public
AppDetails = this.Schedule.getEvents(); |

| Get current view dates | Not applicable | Method: getCurrentViewDates() <br><br>
@ViewChild('schedule') Schedule: ScheduleComponent;<br>public AppDetails =
this.Schedule.getCurrentViewDates(); |

| Get event details | Not applicable | Method: getEventDetails() <br><br> @ViewChild('schedule')
Schedule: ScheduleComponent;<br>public AppDetails =
this.Schedule.getEventDetails(appElement); |

| Get occurrences using event ID | Not applicable | Method: getOccurrencesByID() <br><br>
@ViewChild('schedule') Schedule: ScheduleComponent;<br>public recCollection =
this.Schedule.getOccurrencesByID(1); |

| Get occurrences in the provided date range | Not applicable | Method: getOccurrencesByRange()
<br><br> @ViewChild('schedule') Schedule: ScheduleComponent;<br>public sDate = new
Date(2018, 1, 12); <br> public eDate = new Date(2018, 1, 17); <br> public resCollection =
this.Schedule.getOccurrencesByRange(sDate, eDate);|

Get resource details using index | Not applicable | Method: getResourceByIndex() <br><br>
@ViewChild('schedule') Schedule: ScheduleComponent;<br>public resCollection =
this.Schedule.getResourceByIndex(2); |

```

| Show spinner | Not applicable | **Method:** *showSpinner()*

 @ViewChild('schedule')
Schedule: ScheduleComponent;
this.Schedule.showSpinner() |

| Hide spinner | Not applicable | **Method:** *hideSpinner()*

 @ViewChild('schedule')
Schedule: ScheduleComponent;
this.Schedule.hideSpinner() |

| Check whether the time slot is available | Not applicable | **Method:** *isSlotAvailable()*

@ViewChild('schedule') Schedule: ScheduleComponent;
public sTime = new Date(2018, 1,
14, 09, 30);
 public etime = new Date(2018, 1, 14, 10, 30);
 public resCollection =
this.Schedule.isSlotAvailable(sTime, eTime); |

| Open the event window manually | Not applicable | **Method:** *openEditor()*

@ViewChild('schedule') Schedule: ScheduleComponent;
public td =
document.querySelector(".e-content-table tbody tr td");
public cellDetail =
this.schedule.getCellDetails(td);
 this.Schedule.openEditor(cellDetail); |

| Refresh the scheduler | **Method:** *refresh()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.refresh();
}
} | **Method:** *refresh()*

@ViewChild('schedule') Schedule: ScheduleComponent;
this.Schedule.refresh() |

| Refresh the events | **Method:** *refreshAppointments()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.refreshAppointments();
}
} | **Method:**
refreshEvents()

 @ViewChild('schedule') Schedule:
ScheduleComponent;
this.Schedule.refreshEvents() |

| Refresh the scroller | **Method:** *refreshScroller()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.refreshScroller();
}
} | Not applicable |

| To remove resources dynamically | **Method:** *removeResource()*

 export class
AppComponent {
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
var resID = 1;
this.Schedule.widget.removeResource(resID,
"Owners");
}
} | **Method:** *removeResource()*

 @ViewChild('schedule') Schedule:
ScheduleComponent;
public data = { text: "Paul", id: 3, groupId: 1, color: "#cc99ff"
};
public index = 0;
this.Schedule.removeResource(index, "Owners"); |

| Export scheduler as PDF | **Method:** *exportSchedule()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.exportSchedule("ActionName", null, null);
}
}

| Not Applicable |

| Export scheduler events to ICS | **Method:** *exportSchedule()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.exportSchedule("ActionName","ExportToICS",
null);
}
} | **Method:** *exportToCalendar()*

 @ViewChild('schedule') Schedule:
ScheduleComponent;
this.Schedule.exportToCalendar(fileName); |

| Import scheduler events from ICS | Not Applicable | **Method:** *importToCalendar()*

@ViewChild('schedule') Schedule:

ScheduleComponent;
this.Schedule.importToCalendar(fileContent); |

| Export scheduler appointments in Excel file | **Method:** *exportToExcel()*

 export class
AppComponent {
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.exportToExcel("ActionName", null, true);
}
}
| **Method:** *exportToExcel()*

 @ViewChild('schedule') Schedule:
ScheduleComponent;
let exportValues: ExportOptions = { fields: ['Id', 'Subject', 'StartTime',
'EndTime', 'Location'] };
this.Schedule.exportToExcel(exportValues); |

| Print the scheduler | **Method:** *print()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;

ngAfterViewInit(){
this.Schedule.widget.print();
}
} | Not applicable |

| Filter appointments | **Method:** *filterAppointments()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;
 ngAfterViewInit(){

var filter = [{ field: "Subject", operator: "contains", value: "with", predicate: "or" }];
var
filteredApp = this.Schedule.widget.filterAppointments(filter);
}
} | Not applicable |

| Search appointments | **Method:** *searchAppointments()*

 export class AppComponent
{
@ViewChild('schedule') Schedule: EJComponents<any, any>;
 ngAfterViewInit(){
var
searchApp = this.Schedule.widget.searchAppointments("with");
}
} | Not applicable |

| To edit appointments manually | Not applicable | **Method:** *saveEvent()*

@ViewChild('schedule') Schedule: ScheduleComponent;
public Data: Object[] =
[
{
Id: 1,
Subject: 'Conference',
StartTime: new Date(2018, 1, 12, 9, 0),
EndTime: n
ew Date(2018, 1, 12, 10, 0),
IsAllDay: false};
this.Schedule.saveEvent(Data); |

| Setting work hours | Not applicable | **Method:** *setWorkHours()*

 @ViewChild('schedule')
Schedule: ScheduleComponent;
this.Schedule.setWorkHours([new Date(2017, 9, 5),
'04:00', '08:00']); |

| Scrolling to specific time | Not applicable | **Method:** *scrollTo()*

 @ViewChild('schedule')
Schedule: ScheduleComponent;
this.Schedule.scrollTo('12:00'); |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Fires on the beginning of each scheduler action | **Event:** *actionBegin*

 <ej-schedule
#schedule (actionBegin) = "onActionBegin(\$event)">
</ej-schedule>
 TS

onActionBegin(e: any){} | **Event:** *actionBegin*

<ejs-schedule
(actionBegin)='onActionBegin(\$event)'>
</ejs-schedule>
 TS
onActionBegin(args:
any): void{} |

| Fires on the completion of each scheduler action | **Event:** *actionComplete*

 <ej-schedule
#schedule (actionComplete) = "onActionComplete(\$event)">
</ej-schedule>
 TS

onActionComplete(e: any){} | **Event:** *actionComplete*

<ejs-schedule

(actionComplete)=`'onActionComplete($event)'`>
</ej-schedule>
 TS

`onActionComplete(args: any): void{}` |

| Fires when the scheduler action gets failed | Not applicable | **Event:** *actionFailure*

 <ej-schedule (actionFailure)=`'onActionFailure($event)'`>
</ej-schedule>
 TS

`onActionFailure(args: any): void{}` |

| Fires on appointment hover | **Event:** *appointmentHover*

 <ej-schedule #schedule (appointmentHover) = `"onAppointmentHover($event)"`>
</ej-schedule>
 TS

`onAppointmentHover(e: any){}` | Not applicable |

| Fires before an appointment gets created | **Event:** *beforeAppointmentCreate*

 <ej-schedule #schedule (beforeAppointmentCreate) = `"onBeforeAppointmentCreate($event)"`>
</ej-schedule>
 TS

`onBeforeAppointmentCreate(e: any){}` | **Event:** *actionBegin*

 <ej-schedule (actionBegin)=`'onActionBegin($event)'`>
</ej-schedule>
 TS
`onActionBegin(args: any): void{ if(args.requestType == 'eventCreate') {} }` |

| Fires before an appointment gets edited | **Event:** *beforeAppointmentChange*

 <ej-schedule #schedule (beforeAppointmentChange) = `"onBeforeAppointmentChange($event)"`>
</ej-schedule>
 TS
`onBeforeAppointmentChange(e: any){}` | **Event:** *actionBegin*

 <ej-schedule (actionBegin)=`'onActionBegin($event)'`>
</ej-schedule>
 TS

`onActionBegin(args: any): void{ if(args.requestType == 'eventChange') {} }` |

| Fires before an appointment gets deleted | **Event:** *beforeAppointmentRemove*

 <ej-schedule #schedule (beforeAppointmentRemove) = `"onBeforeAppointmentRemove($event)"`>
</ej-schedule>
 TS

`onBeforeAppointmentRemove(e: any){}` | **Event:** *actionBegin*

 <ej-schedule (actionBegin)=`'onActionBegin($event)'`>
</ej-schedule>
 TS
`onActionBegin(args: any): void{ if(args.requestType == 'eventRemove') {} }` |

| Fires before the context menu opens on scheduler | **Event:** *beforeContextMenuOpen*

 <ej-schedule #schedule (beforeContextMenuOpen) = `"onBeforeContextMenuOpen($event)"`>
</ej-schedule>
 TS

`onBeforeContextMenuOpen(e: any){}` | Not applicable |

| Fires on cell click | **Event:** *cellClick*

 <ej-schedule #schedule (cellClick) = `"onCellClick($event)"`>
</ej-schedule>
 TS
`onCellClick(e: any){}` | **Event:** *cellClick*

 <ej-schedule (cellClick)=`'onCellClick($event)'`>
</ej-schedule>
 TS

`onCellClick(args: any): void{}` |

| Fires on cell double click | **Event:** *cellDoubleClick*

 <ej-schedule #schedule (cellDoubleClick) = `"oncellDoubleClick($event)"`>
</ej-schedule>
 TS

`oncellDoubleClick(e: any){}` | **Event:** *cellDoubleClick*

 <ej-schedule (cellDoubleClick)=`'oncellDoubleClick($event)'`>
</ej-schedule>
 TS

`oncellDoubleClick(args: any): void{}` |

| Fires on cell hover | **Event:** *cellHover*

 <ej-schedule #schedule (cellHover) = `"onCellHover($event)"`>
</ej-schedule>
 TS
`onCellHover(e: any){}` | Not applicable |

| Fires once the scheduler is created | **Event:** *create*

 <ej-schedule #schedule (create) = "onCreate(\$event)">
</ej-schedule>
 TS
onCreate(e: any){}| **Event:** *created*

 <ejs-schedule (created) = 'onCreated(\$event)'>
</ejs-schedule>
 TS
onCreated(args: any): void{} |

| Fires on data binding action | Not applicable | **Event:** *dataBinding*

 <ejs-schedule (dataBinding) = 'onDataBinding(\$event)'>
</ejs-schedule>
 TS
onDataBinding(args: any): void{} |

| Fires after the data is bound to the control | Not applicable | **Event:** *dataBound*

 <ej-schedule (dataBound) = 'onDataBound(\$event)'>
</ej-schedule>
 TS
onDataBound(args: any): void{} |

| Fires once the scheduler is destroyed | **Event:** *destroy*

 <ej-schedule #schedule (destroy) = "onDestroy(\$event)">
</ej-schedule>
 TS
onDestroy(e: any){}| **Event:** *destroyed*

 <ej-schedule (destroyed) = 'onDestroyed(\$event)'>
</ej-schedule>
 TS
onDestroyed(args: any): void{} |

| Fires on event click | **Event:** *appointmentClick*

 <ej-schedule #schedule (appointmentClick) = "onAppointmentClick(\$event)">
</ej-schedule>
 TS
onAppointmentClick(e: any){}| **Event:** *eventClick*

 <ejs-schedule (eventClick) = 'onEventClick(\$event)'>
</ejs-schedule>
 TS
onEventClick(args: any): void{} |

| Fires on event double click | **Event:** *appointmentDoubleClick*

 <ej-schedule #schedule (appointmentDoubleClick) = "onAppointmentDoubleClick(\$event)">
</ej-schedule>
 TS
onAppointmentDoubleClick(e: any){}| Not applicable |

| Fires for keyboard actions | **Event:** *keyDown*

 <ej-schedule #schedule (keyDown) = "onKeyDown(\$event)">
</ej-schedule>
 TS
onKeyDown(e: any){}| Not applicable |

| Fires on context menu item click | **Event:** *menuItemClick*

 <ej-schedule #schedule (menuItemClick) = "onMenuItemClick(\$event)">
</ej-schedule>
 TS
onMenuItemClick(e: any){}| Not applicable |

| Fires on navigation | **Event:** *navigation*

 <ej-schedule #schedule (navigation) = "onNavigation(\$event)">
</ej-schedule>
 TS
onNavigation(e: any){}| **Event:** *navigating*

 <ejs-schedule (navigating) = 'onNavigating(\$event)'>
</ejs-schedule>
 TS
onNavigating(args: any): void{} |

| Fires on popup open | **Event:** *appointmentWindowOpen*

 <ej-schedule #schedule (appointmentWindowOpen) = "onAppointmentWindowOpen(\$event)">
</ej-schedule>
 TS
onAppointmentWindowOpen(e: any){}| **Event:** *popupOpen*

 <ejs-schedule (popupOpen) = 'onPopupOpen(\$event)'>
</ejs-schedule>
 TS
onPopupOpen(args: any): void{} |

| Fires on dragging event | **Event:** *drag*

 <ej-schedule #schedule (drag) = "onDrag(\$event)">
</ej-schedule>
 TS
onDrag(e: any){}| **Event:** *drag*

 <ej-schedule (drag) = 'onDrag(\$event)'>
</ej-schedule>
 TS
onDrag(args: any): void{} |

| Fires on drag start | **Event:** *dragStart*

 <ej-schedule #schedule (dragStart) = "onDragStart(\$event)">
</ej-schedule>
 TS
onDragStart(e: any){}| **Event:** *dragStart*

 <ejs-schedule (dragStart) = 'onDragStart(\$event)'">
</ejs-schedule>
 TS
onDragStart(args: any): void{} |

| Fires on drag stop | **Event:** *dragStop*

 <ej-schedule #schedule (dragStop) = "onDragStop(\$event)">
</ej-schedule>
 TS
onDragStop(e: any){}| **Event:** *dragStop*

 <ejs-schedule (dragStop) = 'onDragStop(\$event)'">
</ejs-schedule>
 TS
onDragStop(args: any): void{} |

| Fires on overflow button click | **Event:** *overflowButtonClick*

 <ej-schedule #schedule (overflowButtonClick) = "onOverflowButtonClick(\$event)">
</ej-schedule>
 TS
onOverflowButtonClick(e: any){}| **Event:** *popupOpen*

 <ejs-schedule (popupOpen) = 'onPopupOpen(\$event)'">
</ejs-schedule>
 TS
onPopupOpen(args: any): void{ if(args.type == 'eventContainer') {} } |

| Fires on overflow button hover | **Event:** *overflowButtonHover*

 <ej-schedule #schedule (overflowButtonHover) = "onOverflowButtonHover(\$event)">
</ej-schedule>
 TS
onOverflowButtonHover(e: any){}| Not applicable |

| Fires when the reminder action takes place | **Event:** *reminder*

 <ej-schedule #schedule (reminder) = "onReminder(\$event)">
</ej-schedule>
 TS
onReminder(e: any){}| Not applicable |

| Fires on resizing event | **Event:** *resize*

 <ej-schedule #schedule (resize) = "onResize(\$event)">
</ej-schedule>
 TS
onResize(e: any){}| **Event:** *resize*

 <ejs-schedule (resize) = 'onResize(\$event)'">
</ejs-schedule>
 TS
onResize(args: any): void{} |

| Fires on resize start | **Event:** *resizeStart*

 <ej-schedule #schedule (resizeStart) = "onResizeStart(\$event)">
</ej-schedule>
 TS
onResizeStart(e: any){}| **Event:** *resizeStart*

 <ejs-schedule (resizeStart) = 'onResizeStart(\$event)'">
</ejs-schedule>
 TS
onResizeStart(args: any): void{} |

| Fires on resize stop | **Event:** *resizeStop*

 <ej-schedule #schedule (resizeStop) = "onResizeStop(\$event)">
</ej-schedule>
 TS
onResizeStop(e: any){}| **Event:** *resizeStop*

 <ejs-schedule (resizeStop) = 'onResizeStop(\$event)'">
</ejs-schedule>
 TS
onResizeStop(args: any): void{} |

| Fires on rendering of every scheduler elements | **Event:** *queryCellInfo*

 <ej-schedule #schedule (queryCellInfo) = "onQueryCellInfo(\$event)">
</ej-schedule>
 TS
onQueryCellInfo(e: any){}| **Event:** *renderCell*

 <ejs-schedule (renderCell) = 'onRenderCell(\$event)'">
</ejs-schedule>
 TS
onRenderCell(args: any): void{} |

| Fires before the event rendering on UI | Not applicable | **Event:** *eventRendered*

 <ejs-schedule (eventRendered) = 'onEventRendered(\$event)'">
</ejs-schedule>
 TS
onEventRendered(args: any): void{} |

You can refer to our [Angular Scheduler](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Scheduler example](#) to know how to present and manipulate data.

How To

Add edit and remove events in Angular Schedule component

CRUD actions can be manually performed on appointments using `addEvent`, `saveEvent` and `deleteEvent` methods as shown below.

Normal event

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #addButtonObj ej2-button cssClass='e-info'
(click)='add()'> Add </button>
<button #editButtonObj ej2-button cssClass='e-info' (click)='edit()'> Edit
</button>
<button #deleteButtonObj ej2-button cssClass='e-info' (click)='delete()'>
Delete </button>
<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Week"></e-view> <e-view option="WorkWeek"></e-view> <e-view
option="Month"></e-view> <e-view option="Day"></e-view> </e-views> </ejs-
schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj', { static: true })
  public scheduleObj?: ScheduleComponent;
  @ViewChild('addButtonObj', { static: true })
  public addButtonObj?: ButtonComponent;
  @ViewChild('editButtonObj', { static: true })
  public editButtonObj?: ButtonComponent;
```

```

@ViewChild('deleteButtonObj', { static: true })
public deleteButtonObj?: ButtonComponent;
public data: object [] = [{
  Id: 3,
  Subject: 'Testing',
  StartTime: new Date(2018, 1, 11, 9, 0),
  EndTime: new Date(2018, 1, 11, 10, 0),
  IsAllDay: false
},{
  Id: 4,
  Subject: 'Vacation',
  StartTime: new Date(2018, 1, 13, 9, 0),
  EndTime: new Date(2018, 1, 13, 10, 0),
  IsAllDay: false
}
]);
public selectedDate: Date = new Date(2018, 1, 15);
public eventSettings: EventSettingsModel = { dataSource: this.data };
add(): void {
  let Data: Object[] = [{
    Id: 1,
    Subject: 'Conference',
    StartTime: new Date(2018, 1, 12, 9, 0),
    EndTime: new Date(2018, 1, 12, 10, 0),
    IsAllDay: false
  },{
    Id: 2,
    Subject: 'Meeting',
    StartTime: new Date(2018, 1, 15, 10, 0),
    EndTime: new Date(2018, 1, 15, 11, 30),
    IsAllDay: false
  }
];
  this.scheduleObj?.addEvent(Data);
  this.addButtonObj?.element.setAttribute('disabled', 'true');
}
edit(): void {
  let data: { [key: string]: Object; } = {
    Id: 3,
    Subject: 'Testing-edited',
    StartTime: new Date(2018, 1, 11, 10, 0),
    EndTime: new Date(2018, 1, 11, 11, 0),
    IsAllDay: false
  };
  this.scheduleObj?.saveEvent(data);
  this.editButtonObj?.element.setAttribute('disabled', 'true');
}
delete(): void {
  this.scheduleObj?.deleteEvent(4);
  this.deleteButtonObj?.element.setAttribute('disabled', 'true');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```

```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Recurrence event

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { DataManager, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #addButtonObj ej2-button cssClass='e-info'
(click)='add()'> Add </button>
<button #editButtonObj ej2-button cssClass='e-info' (click)='edit()'> Edit
</button>
<button #deleteButtonObj ej2-button cssClass='e-info' (click)='delete()'>
Delete </button>
<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Week"></e-view> <e-view option="WorkWeek"></e-view> <e-view
option="Month"></e-view> <e-view option="Day"></e-view> </e-views> </ejs-
schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  @ViewChild('addButtonObj')
  public addButtonObj?: ButtonComponent;
  @ViewChild('editButtonObj')
  public editButtonObj?: ButtonComponent;
  @ViewChild('deleteButtonObj')
  public deleteButtonObj?: ButtonComponent;
  public data: object[] = [{
    Id: 3,
    Subject: 'Testing',
    StartTime: new Date(2018, 1, 11, 9, 0),
```

```

        EndTime: new Date(2018, 1, 11, 10, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=3',
    }, {
        Id: 4,
        Subject: 'Vacation',
        StartTime: new Date(2018, 1, 12, 11, 0),
        EndTime: new Date(2018, 1, 12, 12, 0),
        IsAllDay: false,
        RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2',
    }
    ]];
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: this.data };
    add(): void {
        let Data: Object[] = [{
            Id: 1,
            Subject: 'Conference',
            StartTime: new Date(2018, 1, 15, 9, 0),
            EndTime: new Date(2018, 1, 15, 10, 0),
            IsAllDay: false,
            RecurrenceRule: 'FREQ=DAILY;INTERVAL=1;COUNT=2'
        }];
        this.scheduleObj?.addEvent(Data);
        this.addButtonObj?.element.setAttribute('disabled', 'true');
    }
    edit(): void {
        let data: any[] = new
DataManager(this.scheduleObj?.getCurrentViewEvents()).executeLocal(new
Query().where('RecurrenceID', 'equal', 3));
        data[0].Subject = 'Occurrence edited';
        this.scheduleObj?.saveEvent(data[0], 'EditOccurrence');
        this.editButtonObj?.element.setAttribute('disabled', 'true');
    }
    delete(): void {
        this.scheduleObj?.deleteEvent(4, 'DeleteSeries');
        this.deleteButtonObj?.element.setAttribute('disabled', 'true');
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set default value for event fields in Angular Schedule component

Event window default fields name like Title, Location, etc.. can be customized and default value can be set to Subject field using **default** property which will be added if an appointment is created with empty subject.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  providers: [DayService, WeekService, WorkWeekService, MonthService],
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Week"></e-view> <e-view option="WorkWeek"></e-view> <e-view
option="Month"></e-view> <e-view option="Day"></e-view> </e-views> </ejs-
schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: scheduleData,
    fields: {
      subject: { title: 'Event Name', name: 'Subject', default: 'Add
Name' },
      location: { title: 'Event Location', name: 'Location', default:
'USA' },
      description: { title: 'Summary', name: 'Description' },
      startTime: { title: 'From', name: 'StartTime' },
      endTime: { title: 'To', name: 'EndTime' }
    }
  };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Open event editor manually in Angular Schedule component

Open Editor Window externally

Schedule allows user to manually open the event editor on specific time or on certain events using `openEditor` method as shown below.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button ej2-button cssClass='e-info' (click)='editor()'> Click
to open Editor </button>
  <button ej2-button cssClass='e-info' (click)='eventEditor()'> Click to open
Event Editor </button>
  <ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings" > <e-views> <e-
view option="Week"></e-view> <e-view option="WorkWeek"></e-view> <e-view
option="Month"></e-view> <e-view option="Day"></e-view> </e-views> </ejs-
schedule>`
})
export class AppComponent {
  @ViewChild('scheduleObj')
  public scheduleObj?: ScheduleComponent;
  public selectedDate: Date = new Date(2018, 1, 15);
  public eventSettings: EventSettingsModel = { dataSource: scheduleData };
  editor(): void {
    let cellData: Object = {
      startTime: new Date(2018, 1, 15, 10, 0),
      endTime: new Date(2018, 1, 15, 11, 0),
    };
    this.scheduleObj?.openEditor(cellData, 'Add');
  }
  eventEditor(): void {
    let eventData: Object = {
```

```

        Id: 4,
        Subject: 'Meteor Showers in 2018',
        StartTime: new Date(2018, 1, 14, 13, 0),
        EndTime: new Date(2018, 1, 14, 14, 30)
    };
    this.scheduleObj?.openEditor(eventData, 'Save');
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Open editor window on single click

By default, Scheduler Editor window will open when double clicking the cells or appointments. You can also open the editor window with single click by using `openEditor` method in `eventClick` and `cellClick` events of scheduler and setting false to `showQuickInfo`. The following example shows how to open editor window on single click of cells and appointments.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent, CellClickEventArgs,
EventClickArgs } from '@syncfusion/ej2-angular-schedule';
import { schedulerData } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
[showQuickInfo]="showQuickInfo" (cellClick)="onCellClick($event)"
(eventClick)="onEventClick($event)" > <e-views> <e-view option="Week"></e-
view> <e-view option="WorkWeek"></e-view> <e-view option="Month"></e-view>
<e-view option="Day"></e-view> </e-views> </ejs-schedule>`

```



```

    })
    export class AppComponent {
        @ViewChild('scheduleObj')
        public scheduleObj?: ScheduleComponent;
        public selectedDate: Date = new Date(2021, 7, 15);
        public eventSettings: EventSettingsModel = { dataSource: schedulerData };
        public showQuickInfo: Boolean = false;
        onCellClick(args: CellClickEventArgs): void {
            this.scheduleObj?.openEditor(args, 'Add');
        }
        onEventClick(args: EventClickArgs): void {
            if (!args.event as any).RecurrenceRule {
                this.scheduleObj?.openEditor(args.event, 'Save');
            }
            else {
                this.scheduleObj?.quickPopup.openRecurrenceAlert();
            }
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Prevent date navigation in Angular Schedule component

We can prevent navigation while clicking on the date header by simply removing `e-navigate` class from header cells which can be achieved in the `renderCell` event as shown in the below demo.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel, RenderCellEventArgs, View } from
 '@syncfusion/ej2-angular-schedule';
import { removeClass } from '@syncfusion/ej2-base';
import { schedulerData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
        WeekService,
        WorkWeekService,
        MonthService,
        AgendaService,

```

```

        MonthAgendaService],
standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(renderCell)="onRenderCell($event)" [(currentView)]="currentView" > <e-
views> <e-view option="Week"></e-view> <e-view option="WorkWeek"></e-view>
<e-view option="Month"></e-view> <e-view option="Day"></e-view> </e-views>
</ejs-schedule>`
    })
export class AppComponent {
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    public currentView: View = 'WorkWeek';
    onRenderCell(args: RenderCellEventArgs): void {
        if (args.elementType === "dateHeader") {
            removeClass(args.element.childNodes, "e-navigate");
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Half yearly view in Angular Schedule component

The year view of our scheduler displays all the 365 days and their related appointments of a particular year. You can customize the year view by using the following properties.

- [firstMonthOfYear](#)
- [monthsCount](#)
- [monthHeaderTemplate](#)

In the following code example, you can see how to render only the last six months of a year in the scheduler. To start with the month of June, `firstMonthYear` is set to 6 and `monthsCount` is set to 6 to render only 6 months.

APP.COMPONENT.HTML

```

<ejs-schedule width="100%" height="550px" [selectedDate]="selectedDate"
[views]="views"
[currentView]='currentView' [eventSettings]="eventSettings" [group]='group'
[firstMonthOfYear]="firstMonthOfYear" [monthsCount]="monthsCount"
>
<e-resources>
    <e-resource field="OwnerId" title="Owner" name="Owners"
        [dataSource]="ownerDataSource" textField='OwnerText' idField='Id'
        colorField='OwnerColor'>
    </e-resource>
</e-resources>

```

```

<ng-template #monthHeaderTemplate let-data>
  <div>{{getMonthHeaderText (data.date)}}</div>
</ng-template>
<ng-template #resourceHeaderTemplate let-data>
  <div class='template-wrap'>
    <div class="resource-details">
      <div class="resource-name">{{data.resourceData.OwnerText}}</div>
    </div>
  </div>
</ng-template>
<e-views>
  <e-view option="Year"></e-view>
  <e-view option="TimelineYear" displayName="Horizontal TimelineYear"
[isSelected]="isSelected"></e-view>
  <e-view option="TimelineYear" displayName="Vertical TimelineYear"
orientation="Vertical" [group]="groupSettings"></e-view>
</e-views>
</ejs-schedule>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewEncapsulation } from '@angular/core';
import {
  ScheduleComponent, EventSettingsModel, EventRenderedArgs, YearService,
  TimelineYearService, GroupModel, ResizeService,
  DragAndDropService, MonthAgendaService
} from '@syncfusion/ej2-angular-schedule';
import { resourceData } from '../datasource';
@Component({
  imports: [
    ScheduleModule,
    TimePickerModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService,
    YearService, TimelineYearService, ResizeService,
    DragAndDropService],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./index.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public selectedDate: Date = new Date(2021, 7, 4);
  public firstMonthOfYear: number = 6;
  public monthsCount: number = 6;
}

```

```

public eventSettings: EventSettingsModel = {
    dataSource: resourceData
};
public group: GroupModel = {
    resources: ['Owners']
};
public ownerDataSource: Object[] = [
    { OwnerText: 'Nancy', Id: 1, OwnerColor: '#ffaa00' },
    { OwnerText: 'Steven', Id: 2, OwnerColor: '#f8a398' },
    { OwnerText: 'Robert', Id: 3, OwnerColor: '#7499e1' },
    { OwnerText: 'Smith', Id: 4, OwnerColor: '#f8a398' },
    { OwnerText: 'Michael', Id: 5, OwnerColor: '#f8a398' }
];
currentView: any;
isSelected: any;
groupSettings: any;
views: any;
public getMonthHeaderText(date: Date): string {
    return date.toLocaleString('en-us', { month: 'long' }) + ' ' +
date.getFullYear();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Set different work hours in Angular Schedule component

By default, the work hours of the Scheduler is highlighted based on the start and end values provided within the `workHours` property which remains same for all days. To highlight different work hours range for different days, `setWorkHours` method can be used as follows.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent, WorkHoursModel } from
 '@syncfusion/ej2-angular-schedule';
import { scheduleData } from './datasource';
@Component({
    imports: [

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
        WeekService,

```

```

        WorkWeekService,
        MonthService,
        AgendaService,
        MonthAgendaService],
standalone: true,
selector: 'app-root',
// specifies the template string for the Schedule component
template: `<button ej-button cssClass='e-info' (click)='workHour()'>
Change the work hours </button>
    <ejs-schedule #scheduleObj width='100%' height='550px'
[workHours]="workHours" [selectedDate]="selectedDate"
[eventSettings]="eventSettings" > <e-views> <e-view option="Week"></e-view>
<e-view option="WorkWeek"></e-view> <e-view option="Month"></e-view> <e-view
option="Day"></e-view> </e-views> </ejs-schedule>`
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    public workHours: WorkHoursModel = { start: '9:00', end: '11:00' };
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData };
    workHour(): void {
        let dates: Date[] = [new Date(2018, 1, 15), new Date(2018, 1, 17)];
        this.scheduleObj?.setWorkHours(dates, '11:00', '20:00');
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Show tool tip with delay in Angular Schedule component

In default, the Schedule tooltip is showed without any delay. We can set up the delay to the Schedule tooltip with the help of the Tooltip `openDelay` property.

The preview demonstrates setting up the delay to the Schedule tooltip. You can check the Schedule tooltip with delay here.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { EventSettingsModel, ScheduleComponent } from '@syncfusion/ej2-
angular-schedule';
import { scheduleData } from './datasource';
@Component({
imports: [

```

```

        ScheduleModule,
        ButtonModule
    ],
    providers: [DayService,
                WeekService,
                WorkWeekService,
                MonthService,
                AgendaService,
                MonthAgendaService],
    standalone: true,
    selector: 'app-root',
    // specifies the template string for the Schedule component
    template: `<ejs-schedule #scheduleObj width='100%' height='550px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"
(created)="onCreated($event)">
</ejs-schedule>`
  })
  export class AppComponent {
    @ViewChild('scheduleObj') public scheduleObj?: ScheduleComponent;
    public selectedDate: Date = new Date(2018, 1, 15);
    public eventSettings: EventSettingsModel = { dataSource: scheduleData,
enableTooltip: true };
    public onCreated(eventData: any): void {
      // Assigning the tooltip object to the tooltipObj variable.
      let tooltipObj = (this.scheduleObj?.element as any).ej2_instances[2];
      // Disable the tooltip to follow the mouse pointer position
      tooltipObj.mouseTrail = false;
      // Setting tooltip open delay
      tooltipObj.openDelay = 1000;
      // Setting the position to the tooltip
      tooltipObj.position = "TopCenter";
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Render schedule inside dialog in Angular Schedule component

Render the Schedule while opening the dialog inside the angular `ng-template`.

If you render the Schedule before the dialog is opened it will cause wrong calculations on layout and appointment rendering. To avoid this problem render the Schedule only when the dialog is opened.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { ScheduleAllModule, RecurrenceEditorAllModule } from
 '@syncfusion/ej2-angular-schedule'
import { DialogAllModule } from '@syncfusion/ej2-angular-popups'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'

```

```

import { RouterModule } from '@angular/router'
import { CommonModule } from '@angular/common'
import { Component, ViewChild, Inject } from '@angular/core';
import { DialogComponent } from '@syncfusion/ej2-angular-popups';
import { TimelineViewsService , EventSettingsModel} from '@syncfusion/ej2-
angular-schedule';
@Component({
imports: [ CommonModule, ScheduleAllModule, RecurrenceEditorAllModule,
DialogAllModule, ]
standalone: true,
  selector: 'app-root',
  providers: [TimelineViewsService ],
  // specifies the template string for the Schedule component
  template: `<div id="container">
    <button (click)="onOpenDialog()">Open dialog</button>
    <ejs-dialog id="modalDialog" #modalDialog [closeOnEscape]='false'
[visible]='false'
    [position]='position' [animationSettings]="animationSettings"
height='94%' width='100%' [showCloseIcon]='true'>
      <ng-template #content>
        <div *ngIf="renderSchedule">
          <ejs-schedule #schedule height='448px' width="100%"
[(currentView)]="currentView" [(selectedDate)]="selectedDate"
[eventSettings]="eventSettings">
            <e-views>
              <e-view option="TimelineDay"></e-view>
              <e-view option="TimelineWeek"></e-view>
            </e-views>
          </ejs-schedule>
        </div>
      </ng-template>
    </ejs-dialog>
  </div>`,
})
export class AppComponent {
  @ViewChild('modalDialog') modalDialog: DialogComponent | undefined;
  public animationSettings: Object = { effect: 'SlideTop', duration: 400,
delay: 0 };
  public renderSchedule: boolean = false;
  public selectedDate: Date = new Date(2022, 1, 3);
  public currentView = "TimelineWeek";
  public eventSettings: EventSettingsModel = {
    dataSource:[{
      Id: 1,
      Subject: 'Board Meeting',
      Description: 'Meeting to discuss business goal of 2020.',
      StartTime: new Date(2022, 1, 3, 15, 0),
      EndTime: new Date(2022, 1, 3, 17, 0),
    },
    {
      Id: 2,
      Subject: 'Training session on JSP',
      Description: 'Knowledge sharing on JSP topics.',
      StartTime: new Date(2022, 1, 4, 15, 0),
      EndTime: new Date(2022, 1, 4, 17, 0),
    }
  ],
  };

```

```
position: any;
    public onOpenDialog() {
        this.modalDialog!.show();
        this.renderSchedule = true;
    }
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

[Quick info template in Angular Schedule component](#)

This demo showcases the quick popups for cells and appointments with the customized templates.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxAllModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild, ViewEncapsulation, Inject } from '@angular/core';
import { extend, Internationalization } from '@syncfusion/ej2-base';
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService,
MonthAgendaService, CurrentAction, EventSettingsModel, ResourcesModel, CellClickEventArgs, EJ2Instance, ScheduleComponent } from '@syncfusion/ej2-angular-schedule';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { scheduleData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        DropDownListAllModule,
        TextBoxAllModule,
        ButtonAllModule
    ],
    standalone: true,
    selector: 'app-root',
    providers: [DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService],
    encapsulation: ViewEncapsulation.None,
    // specifies the template string for the Schedule component
    template: `<div class="control-section">
        <ejs-schedule #scheduleObj width='100%' height='600px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings">
        <e-resources>
```



```

        <e-resource field='RoomId' title='Room Type' name='MeetingRoom'
textField='Name' idField='Id'
        colorField='Color' [dataSource]='roomData'>
    </e-resource>
</e-resources>
<!-- Header template -->
<ng-template #quickInfoTemplatesHeader let-data>
    <div class="quick-info-header">
        <div class="quick-info-header-content"
[ngStyle]="getHeaderStyles(data)">
            <div class="quick-info-
title">{{getHeaderTitle(data)}}</div>
            <div class="duration-
text">{{getHeaderDetails(data)}}</div>
        </div>
    </div>
</ng-template>
<!-- Content Template -->
<ng-template #quickInfoTemplatesContent let-data>
    <ng-container [ngTemplateOutlet]="data.elementType == 'cell' ?
cellContent : eventContent"
        [ngTemplateOutletContext]="{data:data}"></ng-container>
</ng-template>
<ng-template #cellContent let-data="data">
    <div class="quick-info-content">
        <div class="e-cell-content">
            <div class="content-area">
                <ejs-textbox #titleObj id="title"
placeholder="Title"></ejs-textbox>
            </div>
            <div class="content-area">
                <ejs-dropdownlist id='eventType' #eventTypeObj
[dataSource]='roomData' [fields]='roomFields'
                placeholder='Choose Type' index="{{0}}"
popupHeight="200px"></ejs-dropdownlist>
            </div>
            <div class="content-area">
                <ejs-textbox #notesObj id="notes"
placeholder="Notes"></ejs-textbox>
            </div>
        </div>
    </div>
</ng-template>
<ng-template #eventContent let-data="data">
    <div class="quick-info-content">
        <div class="event-content">
            <div class="meeting-type-wrap">
                <label>Subject</label>:
                <span>{{data.Subject}}</span>
            </div>
            <div class="meeting-subject-wrap">
                <label>Type</label>:
                <span>{{getEventType(data)}}</span>
            </div>
            <div class="notes-wrap">
                <label>Notes</label>:
                <span>{{data.Description}}</span>
            </div>
        </div>
    </div>
</ng-template>

```

```

        </div>
    </div>
</ng-template>
<!-- Footer Template -->
<ng-template #quickInfoTemplatesFooter let-data>
    <ng-container [ngTemplateOutlet]="data.elementType == 'cell' ?
cellFooter : eventFooter"
        [ngTemplateOutletContext]="{data:data}"></ng-container>
</ng-template>
<ng-template #cellFooter let-data="data">
    <div class="e-cell-footer">
        <button ejs-button id="more-details" cssClass="e-flat"
(click)="buttonClickActions($event)">More
            Details</button>
        <button ejs-button id="add" cssClass="e-flat"
[isPrimary]="true"
            (click)="buttonClickActions($event)">Add</button>
    </div>
</ng-template>
<ng-template #eventFooter let-data="data">
    <div class="e-event-footer">
        <button ejs-button id="delete" cssClass="e-flat"
(click)="buttonClickActions($event)">Delete</button>
        <button ejs-button id="more-details" cssClass="e-flat"
[isPrimary]="true"
            (click)="buttonClickActions($event)">More
            Details</button>
    </div>
</ng-template>
</ejs-schedule>
</div>`,
styleUrls: ['./index.css'],
})
export class AppComponent {
    @ViewChild('scheduleObj')
    public scheduleObj?: ScheduleComponent;
    @ViewChild('eventTypeObj')
    public eventTypeObj?: DropDownListComponent;
    @ViewChild('titleObj')
    public titleObj?: TextBoxComponent;
    @ViewChild('notesObj')
    public notesObj?: TextBoxComponent;
    public eventSettings: EventSettingsModel = { dataSource:
<Object[]>extend([], scheduleData, undefined, true) };
    public selectedDate: Date = new Date(2020, 0, 9);
    public intl: Internationalization = new Internationalization();
    public roomFields: Object = { text: 'Name', value: 'Id' };
    public roomData: Object[] = [
        { Name: 'Jammy', Id: 1, Capacity: 20, Color: '#ea7a57', Type:
'Conference' },
        { Name: 'Tweety', Id: 2, Capacity: 7, Color: '#7fa900', Type: 'Cabin'
},
        { Name: 'Nestle', Id: 3, Capacity: 5, Color: '#5978ee', Type: 'Cabin'
},
        { Name: 'Phoenix', Id: 4, Capacity: 15, Color: '#fec200', Type:
'Conference' },
    ],

```

```

        { Name: 'Mission', Id: 5, Capacity: 25, Color: '#df5286', Type:
'Conference' },
        { Name: 'Hangout', Id: 6, Capacity: 10, Color: '#00bdae', Type:
'Cabin' },
        { Name: 'Rick Roll', Id: 7, Capacity: 20, Color: '#865fcf', Type:
'Conference' },
        { Name: 'Rainbow', Id: 8, Capacity: 8, Color: '#1aaa55', Type:
'Cabin' },
        { Name: 'Swarm', Id: 9, Capacity: 30, Color: '#df5286', Type:
'Conference' },
        { Name: 'Photogenic', Id: 10, Capacity: 25, Color: '#710193', Type:
'Conference' }
    ];
    public getResourceData(data: { [key: string]: Object }): { [key: string]:
Object } {
        // tslint:disable-next-line: deprecation
        const resources: ResourcesModel =
this.scheduleObj!.getResourceCollections()[0];
        const resourceData: { [key: string]: Object } = (resources.dataSource
as Object[]).filter((resource: Object) =>
(resource as { [key: string]: Object; })['Id'] === data['RoomId'])[0] as
{ [key: string]: Object };
        return resourceData;
    }
    public getHeaderStyles(data: { [key: string]: Object }): Object {
        if (data['elementType'] === 'cell') {
            return { 'align-items': 'center', 'color': '#919191' };
        } else {
            const resourceData: { [key: string]: Object } =
this.getResourceData(data);
            return { 'background': resourceData['Color'], 'color': '#FFFFFF'
};
        }
    }
    public getHeaderTitle(data: { [key: string]: Object }): string {
        return (data['elementType'] === 'cell') ? 'Add Appointment' :
'Appointment Details';
    }
    public getHeaderDetails(data: { [key: string]: Date }): string {
        return this.intl.formatDate(data['StartTime'], { type: 'date',
skeleton: 'full' }) + ' (' +
            this.intl.formatDate(data['StartTime'], { skeleton: 'hm' }) + ' -
' +
            this.intl.formatDate(data['EndTime'], { skeleton: 'hm' }) + ')';
    }
    public getEventType(data: { [key: string]: string }): string {
        const resourceData: { [key: string]: Object } =
this.getResourceData(data);
        return resourceData['Name'] as string;
    }
    public buttonClickActions(e: Event) {
        const quickPopup: HTMLElement =
this.scheduleObj?.element.querySelector('.e-quick-popup-wrapper') as
HTMLElement;
        const getSlotData: Function = (): { [key: string]: Object } => {
            const cellDetails: CellClickEventArgs =
this.scheduleObj!.getCellDetails(this.scheduleObj!.getSelectedElements());

```

```

        const addObj: { [key: string]: Object } = {};
        addObj['Id'] = this.scheduleObj!.getEventMaxID();
        addObj['Subject'] = (this.titleObj as TextBoxComponent).value;
        addObj['StartTime'] = new Date(+cellDetails.startTime);
        addObj['EndTime'] = new Date(+cellDetails.endTime);
        addObj['Description'] = (this.notesObj as
TextBoxComponent).value;
        addObj['RoomId'] = (this.eventTypeObj as
DropDownListComponent).value;
        return addObj;
    };
    if ((e.target as HTMLElement).id === 'add') {
        const addObj: { [key: string]: Object } = getSlotData();
        this.scheduleObj!.addEvent(addObj);
    } else if ((e.target as HTMLElement).id === 'delete') {
        const eventDetails: { [key: string]: Object } =
this.scheduleObj!.activeEventData.event as { [key: string]: Object };
        let currentAction: CurrentAction = 'Delete';
        if (eventDetails['RecurrenceRule']) {
            currentAction = 'DeleteOccurrence';
        }
        this.scheduleObj!.deleteEvent(eventDetails, currentAction);
    } else {
        const isCellPopup: boolean =
quickPopup.firstElementChild!.classList.contains('e-cell-popup');
        const eventDetails: { [key: string]: Object } = isCellPopup ?
getSlotData() :
Object };
        let currentAction: CurrentAction = isCellPopup ? 'Add' : 'Save';
        if (eventDetails['RecurrenceRule']) {
            currentAction = 'EditOccurrence';
        }
        this.scheduleObj?.openEditor(eventDetails, currentAction, true);
    }
    this.scheduleObj?.closeQuickInfoPopup();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Enable scroll option on all day section in Angular Schedule component

When you have larger number of appointments in all-day row, it is difficult to view all the appointments properly. In that case you can enable scroller option for all-day row by setting true to `enableAllDayScroll` whereas its default value is false. When setting this property to true, individual scroller for all-day row is enabled when it reaches its maximum height on expanding.

Note: This property is not applicable for Scheduler with height `auto`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component } from '@angular/core';
import { EventSettingsModel } from '@syncfusion/ej2-angular-schedule';
import { generateObject } from './datasource';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<ejs-schedule width='100%' height='550px'
[selectedDate]="selectedDate"
[eventSettings]="eventSettings"
[enableAllDayScroll]="enableAllDayScroll"></ejs-schedule>`
})
export class AppComponent {
  public selectedDate: Date = new Date(2021, 3, 28);
  public eventSettings: EventSettingsModel = {
    dataSource: generateObject(),
  };
  public enableAllDayScroll?: true;
}

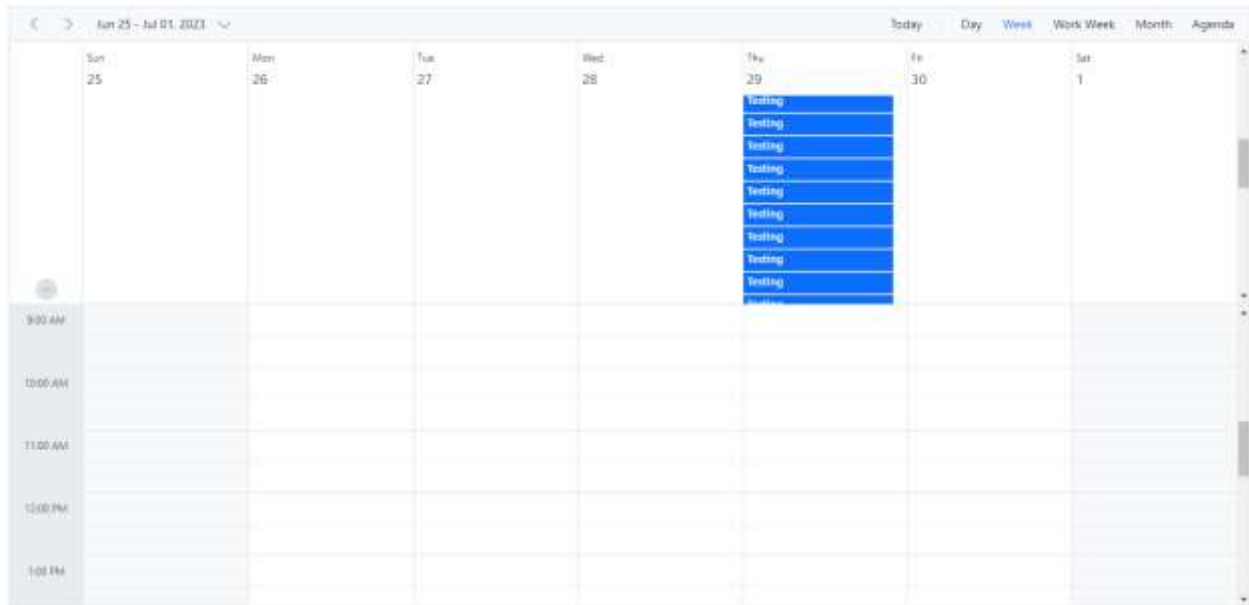
```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```



Manual refresh in Angular Schedule component

Refresh Template

In Scheduler, we can able to refresh the elements of the template alone instead of the entire scheduler by using the [refreshTemplates](#) public method. We can provide an additional option to refresh specific templates alone or all templates together by using this method. The following template names are accepted as an argument to refresh it specifically.

- eventTemplate
- dateHeaderTemplate
- resourceHeaderTemplate
- quickInfoTemplates
- editorTemplate
- tooltipTemplate
- headerTooltipTemplate

In the following code example, you can see how to use the `refreshTemplates` method to refresh multiple templates. Here, we have added the following scheduler templates such as [cellTemplate](#), [dateHeaderTemplate](#), [eventTemplate](#) and [resourceHeaderTemplate](#)

APP.COMPONENT.HTML

```
<div>
  <div style="display: flex;">
    <div style="padding-right: 10px;">
      <button ej-button cssClass='e-info'
      (click)='refreshCellTemplate()'> Refresh
      cellTemplate </button>
    </div>
    <div style="padding-right: 10px;">
      <button ej-button cssClass='e-info'
      (click)='refreshDateHeaderTemplate()'> Refresh
      dateHeaderTemplate </button>
    </div>
  </div>
```

```

        <div style="padding-right: 10px;">
            <button ejs-button cssClass='e-info'
(click)='refreshEventTemplate()'> Refresh
                eventTemplate </button>
        </div>
        <div style="padding-right: 10px;">
            <button ejs-button cssClass='e-info'
(click)='refreshResHeaderTemplate()'> Refresh
                resourceHeaderTemplate </button>
        </div>
        <div style="padding-right: 10px;">
            <button #allTempObj ejs-button cssClass='e-info'
(click)='refreshAllTemplate()'> Refresh All
                Template </button>
        </div>
    </div>
    <div>
        <ejs-schedule #scheduleObj width='auto' height='520px'
[selectedDate]="selectedDate"
        [(currentView)]="currentView" [eventSettings]="eventSettings"
[group]="group" [readonly]='readonly'>
            <ng-template #resourceHeaderTemplate let-data>
                <div class='res-template-wrap'>
                    <div class="resource-image
{{getDoctorImage(data)}}"></div>
                    <div class="resource-details">
                        <div class="resource-
name">{{getDoctorName(data)}}</div>
                        <div class="resource-
designation">{{getDoctorLevel(data)}}</div>
                    </div>
                </div>
            </ng-template>
            <ng-template #dateHeaderTemplate let-data>
                <div class="date-text">{{getDateHeaderText(data.date)}}</div>
                <div [innerHTML]="getWeatherImage(data.date)"></div>
            </ng-template>
            <ng-template #cellTemplate let-data>
                <div class="cell-template-wrap" *ngIf="data.type ==
'workCells'"
                    [innerHTML]="getWorkCellText(data.date)">
                </div>
                <div class="cell-template-wrap" *ngIf="data.type ==
'monthCells'"
                    [innerHTML]="getMonthCellText(data.date)">
                </div>
            </ng-template>
        <e-views>
            <e-view option="Week">
                <ng-template #eventTemplate let-data>
                    <div class='app-template-wrap'
[style.background]="data.SecondaryColor">
                        <div class="subject"
[style.background]="data.PrimaryColor">{{data.Subject}}</div>
                        <div class="time"
[style.background]="data.PrimaryColor">Time:
                            {{getTimeString(data.StartTime)}}

```

```

-
    {{getTimeString(data.EndTime)}}</div>
    <div class="image">
        
    </div>
    <div
class="description">{{data.Description}}</div>
        <div class="footer"
[style.background]="data.PrimaryColor"></div>
    </div>
</ng-template>
</e-view>
<e-view option="Month"></e-view>
<e-view option="TimelineMonth"></e-view>
</e-views>
<e-resources>
    <e-resource field='DoctorId' title='Doctor Name'
name='Doctors' [dataSource]='resourceDataSource'
        textField='text' idField='id' colorField='color'
workDaysField='workDays' startHourField='startHour'
        endHourField='endHour'>
    </e-resource>
</e-resources>
</ejs-schedule>
</div>
</div>

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { CheckBoxAllModule, ButtonModule } from '@syncfusion/ej2-angular-
buttons'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { extend, Internationalization } from "@syncfusion/ej2-base";
import {
    EventSettingsModel, ScheduleComponent, WeekService, MonthService,
    TimelineMonthService, ResizeService,
    DragAndDropService, View, GroupModel, ResourceDetails
} from "@syncfusion/ej2-angular-schedule";
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { webinarData } from './datasource';
@Component({
    imports: [
        ScheduleModule,
        TimePickerModule,
        CheckBoxAllModule,
        ButtonModule
    ],
    standalone: true,

```



```

        selector: 'app-root',
        templateUrl: './app.component.html',
        providers: [WeekService, MonthService, TimelineMonthService,
ResizeService, DragAndDropService],
        styleUrls: ['./index.css'],
        encapsulation: ViewEncapsulation.None
    })
}
export class AppComponent {
    @ViewChild('scheduleObj', { static: true })
    public scheduleObj?: ScheduleComponent;
    public eventSettings: EventSettingsModel = { dataSource: extend([],
webinarData, undefined, true) as Record<string, any>[] };
    public currentView: View = 'Week';
    public readonly = true;
    public selectedDate: Date = new Date(2021, 1, 15);
    public instance: Internationalization = new Internationalization();
    public resourceDataSource: Record<string, any>[] = [
        { text: 'Will Smith', id: 1, color: '#ea7a57', workDays: [1, 2, 4,
5], startHour: '08:00', endHour: '15:00' },
        { text: 'Alice', id: 2, color: 'rgb(53, 124, 210)', workDays: [1, 3,
5], startHour: '08:00', endHour: '17:00' },
        { text: 'Robson', id: 3, color: '#7fa900', startHour: '08:00',
endHour: '16:00' }
    ];
    public group: GroupModel = { resources: ['Doctors'] };
    ImageName: any;
    constructor() {
    }
    public getDoctorName(value: ResourceDetails): string {
        return ((value as ResourceDetails).resourceData) ?
            (value as ResourceDetails).resourceData[(value as
ResourceDetails).resource.textField!] as string
            : value.resourceName as string;
    }
    public getDoctorImage(value: ResourceDetails): string {
        const resourceName: string = this.getDoctorName(value);
        return resourceName.replace(' ', '-').toLowerCase();
    }
    public getDoctorLevel(value: ResourceDetails): string {
        const resourceName: string = this.getDoctorName(value);
        return (resourceName === 'Will Smith') ? 'Cardiologist' :
(resourceName === 'Alice') ? 'Neurologist' : 'Orthopedic Surgeon';
    }
    public getWeatherImage(value: Date): string {
        switch (value.getDay()) {
            case 0:
                return '<div class="weather-text">25°C</div>';
            case 1:
                return '<div class="weather-text">18°C</div>';
            case 2:
                return '<div class="weather-text">10°C</div>';

```

```

        case 3:
            return '<div class="weather-text">16°C</div>';
        case 4:
            return '<div class="weather-text">8°C</div>';
        case 5:
            return '<div class="weather-text">27°C</div>';
        case 6:
            return '<div class="weather-text">17°C</div>';
        default:
            return '';
    }
}

public getDateHeaderText(value: Date): string {
    return this.instance.formatDate(value, { skeleton: 'Ed' });
}

getMonthCellText(date: Date): string {
    if (date.getMonth() === 1 && date.getDate() === 23) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 9) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/get-together.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 13) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    } else if (date.getMonth() === 1 && date.getDate() === 22) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/thanksgiving-day.svg"
/>';
    } else if (date.getMonth() === 1 && date.getDate() === 14) {
        return '<img src=
"https://ej2.syncfusion.com/demos/src/schedule/images/birthday.svg" />';
    }
    return '';
}

getWorkCellText(date: Date): string {
    let weekEnds: number[] = [0, 6];
    if (weekEnds.indexOf(date.getDay()) >= 0) {
        return "<span class='caption'>Weekend</span>";
    }
    return '';
}

public getTimeString(value: Date): string {
    return this.instance.formatDate(value, { skeleton: 'hm' });
}

refreshCellTemplate(): void {
    this.scheduleObj?.refreshTemplates("cellTemplate");
}

refreshDateHeaderTemplate(): void {

```

```

        this.scheduleObj?.refreshTemplates("dateHeaderTemplate");
    }
    refreshEventTemplate(): void {
        this.scheduleObj?.refreshTemplates("eventTemplate");
    }
    refreshResHeaderTemplate(): void {
        this.scheduleObj?.refreshTemplates("resourceHeaderTemplate");
    }
    refreshAllTemplate(): void {
        this.scheduleObj?.refreshTemplates();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Refresh Layout

In Scheduler, we can able to refresh the layout manually without re-render the DOM element by using the [refreshLayout](#) public method. The following example code explains to know how to use the refreshLayout method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ScheduleModule } from '@syncfusion/ej2-angular-schedule'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DayService, WeekService, WorkWeekService, MonthService,
AgendaService, MonthAgendaService } from '@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { ScheduleComponent, EventSettingsModel, View } from '@syncfusion/ej2-
angular-schedule';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    ScheduleModule,
    ButtonModule
  ],
  providers: [DayService,
    WeekService,
    WorkWeekService,
    MonthService,
    AgendaService,
    MonthAgendaService],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the Schedule component
  template: `<button #refButton ej2-button id="refButton" cssClass='e-info'
content="Refresh Layout" (click)="onButtonClick()"></button>`
})

```

```

<ejs-schedule #scheduleObj width='100%' height='520px'
[selectedDate]="selectedDate" [eventSettings]="eventSettings"></ejs-
schedule>`
)})
export class AppComponent {
  @ViewChild("scheduleObj")
  public scheduleObj?: ScheduleComponent;
  @ViewChild("refButton")
  public refButton?: ButtonComponent;
  public selectedDate: Date = new Date(2021, 10, 15);
  public eventSettings: EventSettingsModel = {
    dataSource: [{
      Id: 1,
      Subject: 'Testing',
      StartTime: new Date(2021, 10, 16, 10, 0),
      EndTime: new Date(2021, 10, 16, 12, 0),
      IsAllDay: false
    }, {
      Id: 2,
      Subject: 'Vacation',
      StartTime: new Date(2021, 10, 18, 10, 0),
      EndTime: new Date(2021, 10, 18, 12, 0),
      IsAllDay: false
    }
  ]
}
  public onClick(): void {
    this.scheduleObj?.refreshLayout();
    this.refButton?.element.setAttribute('disabled', 'true');
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Sidebar

Getting started with Angular Sidebar component

This section briefly explains the steps required to create a simple [Link to the Video](#) component, and demonstrates the basic usage of the [Angular Sidebar control](#) in a Angular CLI application.

To get start quickly with Angular Sidebar using CLI and Schematics, you can check on this video:

Prerequisites

To get started with **Sidebar** component, ensure the compatible versions of Angular and Typescript.

- Angular : 4+
- Typescript : 2.6+

Setting up angular project

Angular provides the easiest way to set angular CLI projects using Angular CLI tool.

Install the CLI application globally to your machine by using following command.

```
`sh
```

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

Create a new application

```
ng new syncfusion-angular-app
```

```
,
```

Navigate to the created project folder by using following command.

```
`sh
```

```
cd syncfusion-angular-app
```

```
,
```

Refer [Syncfusion Angular Getting Started](#) section to know more about setting up **angular-cli** project.

Adding Dependencies

Below dependency packages are required in order to use the **Sidebar** component in your application.

```
`javascript
```

```
|-- @syncfusion/ej2-angular-navigations
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-data
```

```
|-- @syncfusion/ej2-angular-base
```

```
|-- @syncfusion/ej2-navigations
```

```
|-- @syncfusion/ej2-inputs
```

```
|-- @syncfusion/ej2-lists
```

```
|-- @syncfusion/ej2-splitbuttons
```

```
|-- @syncfusion/ej2-popups
```

```
|-- @syncfusion/ej2-buttons
```

```
,
```

Installing Syncfusion Sidebar Package

Syncfusion packages are distributed in npm as **@syncfusion** scoped packages. You can get all the Angular Syncfusion package from npm [link](#).

Currently, Syncfusion provides two types of package structures for Angular components,

1. Ivy library distribution package [format](#)
2. Angular compatibility compiler(Angular's legacy compilation and rendering pipeline) package.

Ivy library distribution package

Syncfusion Angular packages(>=20.2.36) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package use the below command.

Add [@syncfusion/ej2-angular-navigations](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-navigations --save
`
```

Angular compatibility compiled package(ngcc)

For Angular version below 12, you can use the legacy (ngcc) package of the Syncfusion Angular components. To download the `ngcc` package use the below.

Add [@syncfusion/ej2-angular-navigations@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-navigations@ngcc --save
`
```

To mention the ngcc package in the `package.json` file, add the suffix `-ngcc` with the package version as below.

```
`bash
@syncfusion/ej2-angular-navigations:"20.2.38-ngcc"
`
```

Note: If the ngcc tag is not specified while installing the package, the Ivy Library Package will be installed and this package will throw a warning.

Adding Styles

To render the Sidebar component, need to import Sidebar and its dependent component's styles as given below in `app.component.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-navigations/styles/material.css';
`
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Adding Sidebar module

After installing the package, the sidebar component module is available to configure into your application from installed syncfusion package.

Refer to the following snippet to import the sidebar module in `app.module.ts` from the `@syncfusion/ej2-angular-navigations`.

```
`typescript
```

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
// Imported syncfusion sidebar module from navigations package
import { SidebarModule } from '@syncfusion/ej2-angular-navigations';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    // Registering EJ2 Sidebar Module
    SidebarModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Adding Syncfusion component

Add the sidebar component by using `<ejs-sidebar>` selector in `template` section of the `app.component.ts` file.

Refer the sidebar component snippet in `app.component.ts` as follows.

```

`ts
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: ` <ejs-sidebar id="default-sidebar" >
<div class="title"> Sidebar content</div>
</ejs-sidebar>
<div>
<div class="title">Main content</div>
<div class="sub-title">
Content goes here.

```

```

</div>
</div>`,
styleUrls: ['app/app.component.css']
})
export class AppComponent {
}
`

```

Run the application

Use the npm run start command to run the application in the browser.

```

`sh
npm start
`

```

The following samples shows the sidebar component in browser.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden">
    <div class="title"> Sidebar content</div>
  </ejs-sidebar>
  <div>
    <div class="title">Main content</div>
    <div class="sub-title">
      Content goes here.
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public onCreated(args: any) {
    (this.sidebar as SidebarComponent).element.style.visibility = '';
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';

```



```
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Note: The ViewChild property need two parameters in **Angular 7+**, use this @ViewChild(ChildDirective,{static: false}) syntax in **Angular 7+**.

Enable backdrop

Enabling the [showBackdrop](#) in the Sidebar component will prevent the main content from user interactions, when it is in expanded state.

Here, DOM elements will not get changed. It only close the main content by covering with black backdrop overlay and focus only the Sidebar in screen. Sidebar can be rendered with specific width by setting [width](#) property.

Note: To achieve a proper **backdrop**, we suggest that you create a wrapper parent container for the div block in which you intend to enable the backdrop. Set the class name of this parent container as the **target** for the Sidebar. Alternatively, you can place an empty div container after the target container.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden" [width]="width"
[showBackdrop]="showBackdrop" [closeOnDocumentClick]="closeOnDocumentClick">
    <div class="title"> Sidebar content</div>
    <div class="sub-title">
      Click the button to close the Sidebar.
    </div>
    <div class="center-align">
      <button ej-button id="close" (click)="closeClick()"
class="e-btn close-btn">Close Sidebar</button>
    </div>
  </ejs-sidebar>
<div>
  <div class="title">Main content</div>
  <div class="sub-title"> Click the button to open/close
the Sidebar.</div>
  <div style="padding:20px" class="center-align">
    <button ej-button id="toggle" class="e-btn e-info"
(click)="toggleClick()">Toggle Sidebar</button>
  </div>
</div>`
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public showBackdrop: boolean = true;
  public type: string = 'Push';
  public width: string = '280px';
```

```

public closeOnDocumentClick: boolean = true;
public onCreate(args: any) {
    (this.sidebar as SidebarComponent).element.style.visibility = '';
}
closeClick(): void {
    this.sidebar?.hide();
};
toggleClick():void{
    this.sidebar?.show();
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Position

Positioning the Sidebar to the right or left of the main content can be achieved by using the [position](#) property. If the position is not set, the Sidebar will expand from the left to the body element. [enablePersistence](#) will persist the component's state between page reloads. [change](#) event will be triggered when the state(expand/collapse) of the component is changed.

Note: Add the required Button and Radio Button component style dependency to **app.component.css**.

APP.COMPONENT.TS

```

import { ButtonModule, RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
import { ButtonComponent, RadioButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [SidebarModule, ButtonModule, RadioButtonModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar [type]='type'
[enablePersistence]='enablePersistence' (created)="onCreated($event)"
style="visibility: hidden" [target]='target'>
      <div class="title"> Sidebar content</div>
      <div class="sub-title">
        Click the button to close the Sidebar.
      </div>
      <div class="center-align">
        <button ej-button id="close"
(click)="closeClick()" class="e-btn close-btn">Close Sidebar</button>
      </div>
    </ejs-sidebar>

```

```

        <div id="head">
            <button ej-button id="toggle" #togglebtn
cssClass="e-flat" iconCss="e-icons burg-icon" isToggle="true"
                (click)="btnClick()" content="Open"></button>
        </div>
        <div id="maincontent" class="content">
            <div>
                <div class="title">Main content</div>
                <div class="rows">
                    <div class="row">
                        <ejs-radiobutton id='left' #radio
label="Left" name="position" checked="true"
                        (change)="changeHandler($event)"></ejs-radiobutton>
                    </div>
                    <div class="row">
                        <ejs-radiobutton id='right' #radio
label="Right" name="position" (change)="changeHandler($event)">
                        </ejs-radiobutton>
                    </div>
                </div>
            </div>
        </div>`
    })
    export class AppComponent {
        @ViewChild('sidebar') sidebar?: SidebarComponent;
        public type: string = 'Push';
        public target: string = 'content';
        public enablePersistence: boolean = true;
        @ViewChild('togglebtn')
        public togglebtn?: ButtonComponent;
        public onCreated(args: any) {
            (this.sidebar as SidebarComponent).element.style.visibility = '';
        }
        btnClick() {
            if ((this.togglebtn as
ButtonComponent).element.classList.contains('e-active')) {
                (this.togglebtn as ButtonComponent).content = 'Open';
                this.sidebar?.hide();
            } else {
                (this.togglebtn as ButtonComponent).content = 'Close';
                this.sidebar?.show();
            }
        }
        closeClick() {
            this.sidebar?.hide();
            (this.togglebtn as ButtonComponent).element.classList.remove('e-
active');
            (this.togglebtn as ButtonComponent).content = 'Open'
        }
        @ViewChild('radio')
        public radiobutton?: RadioButtonComponent;
        public changeHandler(args: any): void {
            (this.sidebar as SidebarComponent).position =
            (args.event.target.ej2_instances[0].label == "Left") ? "Left" : "Right";
        }
    }

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Animate

Animation transitions can be set while expanding or collapsing the Sidebar using the [animate](#) property. By default, [animate](#) property is set to true. [enableRTL](#) will display the sidebar in the right-to-left direction.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden" [animate]="animate"
[enableRtl]="enableRtl">
      <div class="title"> Sidebar content</div>
      <div class="sub-title">
        Click the button to close the Sidebar
      </div>
      <div class="center-align">
        <button ej2-button id="close" (click)="closeClick()"
class="e-btn close-btn">Close Sidebar</button>
      </div>
    </ejs-sidebar>
    <div>
      <div class="title">Main content</div>
      <div class="sub-title"> Click the button to open/close
the Sidebar.</div>
      <div style="padding:20px" class="center-align">
        <button ej2-button id="toggle" class="e-btn e-info"
(click)="toggleClick()">Toggle Sidebar</button>
      </div>
    </div>`
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public animate: boolean = false;
  public enableRtl: boolean = true;
  public type: string = 'Push';
  public onCreated(args: any) {
    (this.sidebar as SidebarComponent).element.style.visibility = '';
  }
}
```

```

    closeClick(): void {
        this.sidebar?.hide();
    };
    toggleClick():void{
        this.sidebar?.toggle();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Close on document click

Sidebar can be closed on document click by setting [closeOnDocumentClick](#) to true. If this property is not set, the Sidebar will not close on document click since its default value is false. Sidebar can be kept opened during rendering using [isOpen](#) property.

APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden"
[closeOnDocumentClick]="closeOnDocumentClick" [isOpen]="isOpen">
    <div class="title"> Sidebar content</div>
</ejs-sidebar>
<div>
    <div class="title">Main content</div>
    <div class="sub-title"> Click the button to open the
Sidebar.</div>
    <div style="padding:20px" class="center-align">
        <button ej2-button id="toggle" class="e-btn e-info"
(click)="toggleClick()">Open Sidebar</button>
    </div>
</div>`
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public isOpen: boolean = true;
  public closeOnDocumentClick: boolean = true;
  public type: string = 'Push';
  public onCreated(args: any) {
    (this.sidebar as SidebarComponent).element.style.visibility = '';
  }
}

```

```
toggleClick():void{
    this.sidebar?.show();
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Enable gestures

Expand or collapse the Sidebar while swiping in touch devices using [enableGestures](#) property. By default, [enableGestures](#) is set to true.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden"
[enableGestures]="enableGestures" >
    <div class="title"> Sidebar content</div>
    <div class="sub-title">
        Click the button to close the Sidebar
    </div>
    <div class="center-align">
        <button ejs-button id="close" (click)="closeClick()"
class="e-btn close-btn">Close Sidebar</button>
    </div>
</ejs-sidebar>
<div>
    <div class="title">Main content</div>
    <div class="sub-title"> Click the button to open/close
the Sidebar.</div>
    <div style="padding:20px" class="center-align">
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="toggleClick()">Toggle Sidebar</button>
    </div>
</div>`
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public enableGestures: boolean = false;
  public type: string = 'Push';
  public onCreated(args: any) {
```

```

        (this.sidebar as SidebarComponent).element.style.visibility = '';
    }
    closeClick(): void {
        this.sidebar?.hide();
    };
    toggleClick():void{
        this.sidebar?.toggle();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

You can refer to our [Angular Sidebar](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Sidebar example](#) that shows how to render the Sidebar in Angular.

See Also

- [Sidebar with navigation menu](#)
- [Sidebar responsive panel](#)
- [Sidebar with listView](#)
- [Initialize sidebar using systemjs](#)
- [Usecase sample](#)

Custom context in Angular Sidebar component

Sidebar has a flexible option to make it initialize, target to any HTML element alongside of the main content of a web page.

By default, it initialize target to the body element. [target](#) property allows users to set target element to initialize the Sidebar inside any HTML container element apart from the body element.

If required, [zIndex](#) can be set when sidebar act as overlay type.

APP.COMPONENT.TS

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
import { ButtonComponent } from "@syncfusion/ej2-angular-buttons";
@Component({
  imports: [SidebarModule, ButtonModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar [type]='type'
[target]='target' (created)="onCreated($event)" style="visibility: hidden">
                <div class="title"> Sidebar content</div>

```

```

        <div class="sub-title">
            Click the button to close the Sidebar.
        </div>
        <div class="center-align">
            <button ejs-button id="close" (click)="closeClick()"
class="e-btn close-btn">Close Sidebar</button>
        </div>
    </ejs-sidebar>
    <div id="head">
        <button ejs-button id="toggle" #togglebtn cssClass="e-
flat" iconCss="e-icons burg-icon" isToggle="true"
            (click)="btnClick()" content="Open"></button>
    </div>
    <div id="maincontent" class="content">
        <div>
            <div class="title">Main content</div>
            <div class="center-align">
                content goes here.
            </div>
        </div>
    </div>`
    })
    export class AppComponent {
        @ViewChild('sidebar') sidebar?: SidebarComponent;
        public type: string = 'Push';
        public target: string = '.content';
        @ViewChild('togglebtn')
        public togglebtn?: ButtonComponent;
        public onCreate(args: any) {
            (this.sidebar as SidebarComponent).element.style.visibility = '';
        }
        btnClick(){
            if((this.togglebtn as ButtonComponent).element.classList.contains('e-
active')){
                (this.togglebtn as ButtonComponent).content = 'Open';
                this.sidebar?.hide();
            }
            else{
                (this.togglebtn as ButtonComponent).content = 'Close';
                this.sidebar?.show();
            }
        }
        closeClick() {
            this.sidebar?.hide();
            (this.togglebtn as ButtonComponent).element.classList.remove('e-
active');
            (this.togglebtn as ButtonComponent).content = 'Open'
        }
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```


See Also

- [Angular routing](#)

Types in Angular Sidebar component

The Sidebar component's expand behaviour can be modified based on the purpose of use.

Expanding types of Sidebar

The Sidebar can be set to initialize based on four different types that are consistent with the main component as explained below. When `dataBind` is invoked, this applies the pending property changes immediately to the component.

Item	Description
-----	-----
Over	Sidebar floats over the main content area.
Push	Sidebar pushes the main content area to appear side-by-side, and shrinks the main content within the screen width.
Slide	Sidebar translates the x and y positions of main content area based on the Sidebar width. The main content area will not be adjusted within the screen width.
Auto	Sidebar with <code>Over</code> type in mobile resolution, and <code>Push</code> type in other higher resolutions.

`Auto` is the default expand mode.

In the following sample, the types of Sidebar are demonstrated.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule, RadioButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
import { ButtonComponent, RadioButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    SidebarModule, ButtonModule, RadioButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar [type]='type'
[target]='target' (created)="onCreated($event)" style="visibility: hidden">
      <div class="title"> Sidebar content</div>
      <div class="sub-title">
        Click the button to close the Sidebar.
      </div>
    `
})
```

```

        <div class="center-align">
            <button ejs-button id="close"
(click)="closeClick()" class="e-btn close-btn">Close Sidebar</button>
        </div>
    </ejs-sidebar>
    <div>
        <div id="head">
            <button ejs-button id="toggle" #togglebtn
cssClass="e-flat" iconCss="e-icons burg-icon" isToggle="true"
(click)="btnClick()" content="Open"></button>
        </div>
        <div id="maincontent" class="content">
            <div>
                <div class="title">Main content</div>
                <div class="rows">
                    <div class="row">
                        <ejs-radiobutton id='push' #radio
label="Push" name="type" checked="true"
(change)="changeHandler($event)"></ejs-radiobutton>
                    </div>
                    <div class="row">
                        <ejs-radiobutton id='slide' #radio
label="Slide" name="type" (change)="changeHandler($event)"></ejs-radiobutton>
                    </div>
                    <div class="row">
                        <ejs-radiobutton id='over' #radio
label="Over" name="type" (change)="changeHandler($event)"></ejs-radiobutton>
                    </div>
                    <div class="row">
                        <ejs-radiobutton id='auto' #radio
label="Auto" name="type" (change)="changeHandler($event)"></ejs-radiobutton>
                    </div>
                </div>
            </div>
        </div>
    </div>`
    })
    export class AppComponent {
        @ViewChild('sidebar') sidebar?: SidebarComponent;
        @ViewChild('togglebtn') togglebtn?: ButtonComponent;
        public type: string = 'Push';
        public target: string = '.content';
        public onCreate(args: any) {
            (this.sidebar as SidebarComponent).element.style.visibility = '';
        }
        btnClick() {
            if((this.togglebtn as ButtonComponent).element.classList.contains('e-
active')) {
                (this.togglebtn as ButtonComponent).content = 'Open';
                this.sidebar?.hide();
            }
            else {
                (this.togglebtn as ButtonComponent).content = 'Close';
                this.sidebar?.show();
            }
        }
        closeClick() {

```

```

        this.sidebar?.hide();
        (this.togglebtn as ButtonComponent).element.classList.remove('e-
active');
        (this.togglebtn as ButtonComponent).content = 'Open'
    }
    // change the togglebtn state
    changestate() {
        if((this.sidebar as SidebarComponent).type == 'Auto'){
            (this.togglebtn as ButtonComponent).element.classList.add('e-
active');
            (this.togglebtn as ButtonComponent).content = 'close'
        }
        else{
            (this.togglebtn as ButtonComponent).element.classList.remove('e-
active');
            (this.togglebtn as ButtonComponent).content = 'Open';
        }
    }
    changeHandler(args: any) : void {
        if(args.event.target.ej2_instances[0].label == 'Over') {
            (this.sidebar as SidebarComponent).type = 'Over';
        } else if (args.event.target.ej2_instances[0].label == 'Push') {
            (this.sidebar as SidebarComponent).type = 'Push';
        } else if (args.event.target.ej2_instances[0].label == 'Slide') {
            (this.sidebar as SidebarComponent).type = 'Slide';
        } else {
            (this.sidebar as SidebarComponent).type = 'Auto';
        }
        this.changestate();
        this.sidebar?.dataBind();
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

See Also

- [How to add sidebar with custom animation](#)
- [How to add multiple sidebar](#)

Docking sidebar in Angular Sidebar component

Dock state of the Sidebar reserves some space on the page that always remains in a visible, when the Sidebar is in collapsed state. It is used to show the short form of a content like icons alone instead of lengthy text. To achieve this , set [enableDock](#) as true along with required [dockSize](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'

```

```

import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    SidebarModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar #dockBar id="dockSidebar"
[enableDock]='enableDock' [width]='width' [dockSize]='dockSize'>
      <div class="dock">
        <ul>
          <li class="sidebar-item" id="toggle"
(click)="toggleClick()">
              <span class="e-icons expand"></span>
              <span class="e-text" title="menu">Menu</span>
            </li>
          <li class="sidebar-item">
              <span class="e-icons home"></span>
              <span class="e-text" title="home">Home</span>
            </li>
          <li class="sidebar-item">
              <span class="e-icons profile"></span>
              <span class="e-text"
title="profile">Profile</span>
            </li>
          <li class="sidebar-item">
              <span class="e-icons info"></span>
              <span class="e-text" title="info">Info</span>
            </li>
          <li class="sidebar-item">
              <span class="e-icons settings"></span>
              <span class="e-text"
title="settings">Settings</span>
            </li>
          </ul>
        </div>
      </ejs-sidebar>
      <div id="main-content container-fluid col-md-12 ">
        <div class="title">Main content</div>
        <div class="sub-title"> Click the expand icon to open and
collapse icons to close the Sidebar.</div>
      </div>
      <div>
      </div>`
  })
export class AppComponent {
  @ViewChild('dockBar') dockBar?: SidebarComponent;
  public enableDock: boolean = true;
  public width: string = '220px';
  public dockSize: string = '72px';
  toggleClick() {
    this.dockBar?.toggle();
  }
}

```

```
}

```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

See Also

- [How to add sidebar navigation](#)

Auto close in Angular Sidebar component

The Sidebar often behaves differently on mobile display and differently on desktop display.

It has an effective feature that offers to set it in opened or closed state depending on the specified screen resolution.

This functionality can be achieved through [mediaQuery](#) property that allows you to set the Sidebar in an expanded state or collapsed state only in user-defined resolution.

window.matchMedia() method returns the object of parsed **media query** string.

The value of matchMedia() can be any of the media features of CSS @media rule. For example, min-width and max-width.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
@Component({
  imports: [
    SidebarModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar [width]="width"
[mediaQuery]= "mediaQuery" [closeOnDocumentClick]='colseOnDocumentClick'>
    <div class="title"> Sidebar content</div>
  </ejs-sidebar>
  <div>
    <div class="title">Main content</div>
    <div class="sub-title">
      Sidebar will collapse and expand in based on screen
      resolution automatically
    </div>
  </div>`
})
export class AppComponent {
  public width: string = '280px';
```

```
public mediaQuery: object = window.matchMedia('(min-width: 600px)');
colseOnDocumentClick: any;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

- In the following sample, the Sidebar will be in expanded state only in resolution below 400px.

max-width is one of media feature which represents maximum width of display area such as width of the view port.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { SidebarModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
@Component({
  imports: [
    SidebarModule
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: `<ejs-sidebar id="default-sidebar" #sidebar [width]="width"
[mediaQuery]= "mediaQuery" [closeOnDocumentClick]='colseOnDocumentClick'>
    <div class="title"> Sidebar content</div>
  </ejs-sidebar>
  <div>
    <div class="title">Main content</div>
    <div class="sub-title">
      Sidebar will collapse and expand in based on screen
      resolution automatically
    </div>
  </div>`
})
export class AppComponent {
  public width: string = '280px';
  public mediaQuery: object = window.matchMedia('(max-width: 400px)');
  colseOnDocumentClick: any;
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Style in Angular Sidebar component

The following content provides the exact CSS structure that can be used to modify the component's appearance based on the user's preference.

Customizing the sidebar

Use the below CSS to customize the sidebar root element.

```
`css
.e-sidebar {
background: #898b2b
}
`
```

Customizing the sidebar based on the positions

Use the below CSS to customize the left positioned sidebar.

```
`css
.e-sidebar.e-left {
border-right: 2px solid red;
}
`
```

Use the below CSS to customize the right positioned sidebar.

```
`css
.e-sidebar.e-right {
border-left: 2px solid red;
}
`
```

Customizing the sidebar based on the active state

Use the below CSS to customize the open state of the left positioned sidebar.

```
`css
.e-sidebar.e-left.e-open {
transition: transform 2.5s ease;
}
`
```

Use the below CSS to customize the open state of the right positioned sidebar.

```
`css
.e-sidebar.e-right.e-open {
```

```
transition: transform 2.5s ease;
}
```

Use the below CSS to customize the closed state of the left positioned sidebar.

```
`css
.e-sidebar.e-left.e-transition.e-close {
transition: transform 2.5s ease, visibility 1200ms;
}
```

Use the below CSS to customize the closed state of the right positioned sidebar.

```
`css
.e-sidebar.e-right.e-transition.e-close {
transition: transform 2.5s ease, visibility 1200ms;
}
```

Customizing the sidebar with dock state

When you enable the Dock support, the "e-dock" class will be added to the root element. Based on that class, you can also customize all the above stated customization. Use the following CSS to customize the sidebar element with a dock state.

```
`css
.e-sidebar.e-dock {
background: #2d323e;
}
```

Customizing the different types of sidebar

Use the below CSS to customize the auto type sidebar.

```
`css
.e-sidebar.e-left.e-auto {
background-color: pink;
}
```

Use the below CSS to customize the push type sidebar.

```
`css
.e-sidebar.e-left.e-push {
```



```
background-color: beige;
}
```

Use the below CSS to customize the over type sidebar.

```
`css
.e-sidebar.e-left.e-over {
background-color: aqua;
}
```

Use the below CSS to customize the slide type sidebar.

```
`css
.e-sidebar.e-left.e-slide {
background-color: green;
}
```

[Customizing the backdrop of the sidebar](#)

Use the below CSS to customize the backdrop of the sidebar.

```
`css
.e-sidebar-overlay {
background-color: aqua;
}
```

[Customizing the sidebar in the RTL direction](#)

When you enable the RTL (right to left direction) support, the "e-rtl" class will be added to the root element. Based on that class, you can also customize all the above stated customization. Use the following CSS to customize the sidebar element in the RTL (right to left direction) mode.

```
`css
.e-sidebar.e-left.e-rtl {
background-color: antiquewhite;
}
```

[Prevent the animation transition for the Sidebar component](#)

Use the below CSS to prevent the animation transition for the Sidebar component.

```
`css
.e-sidebar-context .e-content-animation {
```

```

transition: none !important;
transform: none !important;
}
`

```

Accessibility in Angular Sidebar component

The Sidebar 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 Sidebar 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 Sidebar component followed the [WAI-ARIA](#) patterns to meet accessibility standards. By default, the Sidebar utilizes the [complementary](#) role, with the option to modify the ARIA role based on provided attributes to the root element, depending on the specific use case.

If there are multiple complementary landmark roles or aside elements in a document, it is important to provide a label for each landmark using the `aria-label` attribute. Alternatively, if the aside has a descriptive title, it can be referenced using the `aria-labelledby` attribute. This label will help assistive technology users quickly understand the purpose of each landmark.

For optimal accessibility, it is recommended to incorporate a trigger component that is navigable via a keyboard, such as a button. If this is not feasible, inclusion of the `tabIndex` attribute becomes necessary. Furthermore, explicit handling of the `aria-expanded` attribute is required for the trigger element.

Keyboard interaction

The Sidebar component does not have any inbuilt keyboard interaction support. However, you can utilize the keyboard interactions of focusable elements within the Sidebar component.

Ensuring accessibility

The Sidebar component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Sidebar component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Sidebar component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Initialize sidebar using `systemjs` in Angular Sidebar component

Sidebar can also be initialized using `SystemJS` as follows

Installation and configuration

- To setup basic `Angular` sample use the following commands.

```
`sh
git clone https://github.com/angular/quickstart.git quickstart
cd quickstart
npm install
`
```

For more information, refer to [Angular sample setup](#).

- Install Syncfusion Sidebar packages using the below command.

```
`sh
npm install @syncfusion/ej2-angular-navigations --save
`
```

The above package installs Sidebar dependencies which are required to render the component in an Angular environment.

- Syncfusion `ej2-angular-navigations` packages need to be mapped in `systemjs.config.js` configuration file.

```
`javascript
/

• System configuration for Angular samples
• Adjust as necessary for your application needs.

*/
(function (global) {
System.config({
paths: {
// paths serve as alias
'npm:': 'node_modules/',
"syncfusion:": "node_modules/@syncfusion/", // syncfusion alias
},
// map tells the System loader where to look for things
map: {
// our app is within the app folder
'app': 'app',
// angular bundles
'@angular/core': 'npm:@angular/core/bundles/core.umd.js',
'@angular/common': 'npm:@angular/common/bundles/common.umd.js',
'@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
'@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',
'@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',
```

```

 '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
 '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
 '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
 // syncfusion bundles
 "@syncfusion/ej2-base": "syncfusion:ej2-base/dist/ej2-base.umd.min.js",
 "@syncfusion/ej2-data": "syncfusion:ej2-data/dist/ej2-data.umd.min.js",
 "@syncfusion/ej2-navigations": "syncfusion:ej2-navigations/dist/ej2-navigations.umd.min.js",
 "@syncfusion/ej2-inputs": "syncfusion:ej2-inputs/dist/ej2-inputs.umd.min.js",
 "@syncfusion/ej2-lists": "syncfusion:ej2-lists/dist/ej2-lists.umd.min.js",
 "@syncfusion/ej2-popups": "syncfusion:ej2-popups/dist/ej2-popups.umd.min.js",
 "@syncfusion/ej2-buttons": "syncfusion:ej2-buttons/dist/ej2-buttons.umd.min.js",
 "@syncfusion/ej2-splitbuttons": "syncfusion:ej2-splitbuttons/dist/ej2-splitbuttons.umd.min.js",
 "@syncfusion/ej2-angular-base": "syncfusion:ej2-angular-base/dist/ej2-angular-base.umd.min.js",
 "@syncfusion/ej2-angular-navigations": "syncfusion:ej2-angular-navigations/dist/ej2-angular-navigations.umd.min.js",
 // other libraries
 'rxjs': 'npm:rxjs',
 'angular-in-memory-web-api': 'npm:angular-in-memory-web-api/bundles/in-memory-web-api.umd.js'
 },
 // packages tells the System loader how to load when no filename and/or no extension
 packages: {
   app: {
     defaultExtension: 'js',
     meta: {
       './*.js': {
         loader: 'systemjs-angular-loader.js'
       }
     }
   },
   rxjs: {
     defaultExtension: 'js'
   }
 }

```

```
});
})(this);
`
```

To render the Sidebar component, need to import Sidebar and its dependent component's styles as given below in `style.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-navigations/styles/material.css';
`
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Create a simple Sidebar

Refer the following code to include the Sidebar in application .

- Create an `Angular` component with Sidebar. Add the following sidebar template in component template of

`app.component.ts`

```
`HTML
<ejs-sidebar id="default-sidebar" >
<div class="title"> Sidebar content</div>
</ejs-sidebar>
<div>
<div class="title">Main content</div>
<div class="sub-title">
Content goes here.
</div>
</div>
`
```

- Create an `Angular` module and include the above Sidebar component.
- In the module, declare the Component and Directives required to render the Sidebar.
- Bootstrap the application with the above module.

Refer to the following snippet to import the sidebar module in `app.module.ts` from the `@syncfusion/ej2-angular-navigations`.

```
`Typescript
```

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { SidebarModule } from '@syncfusion/ej2-angular-navigations';

@NgModule({
  imports: [SidebarModule, BrowserModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
`
```

`systemjs.config.js` file should be configured as described in the [Installation and configuration](#) section.

Run the application

Use the npm run start command to run the application in the browser.

```
`sh
npm start
`
```

The following samples shows the sidebar component in browser.

APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar
(created)="onCreated($event)" style="visibility: hidden">
    <div class="title"> Sidebar content</div>
  </ejs-sidebar>
  <div>
    <div class="title">Main content</div>
    <div class="sub-title">
      Content goes here.
    </div>
  </div> `
})
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public onCreated(args: any) {
```

```

        (this.sidebar as SidebarComponent).element.style.visibility = '';
    }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Initialize the sidebar listview in Angular Sidebar component

Any HTML element can be placed in the Sidebar content area. Sidebar supports all types of HTML structures like **TreeView**, **ListView**, etc.

In the following example, Sidebar has been rendered with **ListView** component in its content area.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ListViewModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="default-sidebar" #sidebar [width]="width">
    <div class="title1">Menu</div>
    <div class="closebtn">
      <button ej-button id="close" class="e-btn
close-btn" (click)="closeClick()">
        <span id="innerclose" class="e-icons
close-icon"></span>
      </button>
    </div>
    <div id="listcontainer">
      <ejs-listview id='list'
[dataSource]='data'></ejs-listview>
    </div>
    <div class="sub-title">
      ListView component is rendered in the
sidebar container.
    </div>
  </ejs-sidebar>
</div>
<div>
  <div>
    <div class="title2">Main content</div>
    <div class="sub-title"> Click the button to
open/close the Sidebar.
    <div style="padding:20px" class="center-
align">

```



```

<button ej-button id="toggle" class="e-
btn e-info" (click)="toggleClick()">Toggle Sidebar</button>
</div>
</div>
</div>`
</div>`
}))
export class AppComponent {
  @ViewChild('sidebar') sidebar?: SidebarComponent;
  public width: string = '100%';
  public type: string = 'Over';
  public data: Object[] = [
    { text: 'Home', id: 'list-02' },
    { text: 'Offers', id: 'list-03' },
    { text: 'Support', id: 'list-04' },
    { text: 'Logout', id: 'list-06' }
  ];
  toggleClick() {
    this.sidebar?.toggle();
  }
  closeClick() {
    this.sidebar?.hide();
  }
}
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Open and close the sidebar in Angular Sidebar component

Opening and closing the Sidebar can be achieved with built-in public methods.

- [show\(\)](#): Method to open the Sidebar.
- [hide\(\)](#): Method to close the Sidebar.
- [toggle\(\)](#): Method to toggle between open and close states of the Sidebar.

In the following sample, toggle method has been used to show or hide the Sidebar on button click.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],

```

```

    template: `
      <ejs-sidebar id="default-sidebar" #sidebar
      (open)="open($event)" (close)="close($event)">
        <div class="title"> Sidebar content</div>
        <div class="sub-title">
          Click the button to close the Sidebar.
        </div>
        <div class="center-align">
          <button ejs-button id="close"
            (click)="closeClick()" class="e-btn close-btn">Close Sidebar</button>
        </div>
      </ejs-sidebar>
      <div>
        <div>
          <div class="title">Main content</div>
          <div class="sub-title"> Click the button to Open
the Sidebar.
          <div style="padding:20px" class="center-
align">
            <button ejs-button id="open" class="e-btn
e-info" (click)="openClick()">Open Sidebar</button>
          </div>
        </div>
        <div class="sub-title"> Click the button to
open/close the Sidebar.
        <div style="padding:20px" class="center-
align">
          <button ejs-button id="toggle" class="e-
btn e-info" (click)="toggleClick()">Toggle Sidebar</button>
        </div>
      </div>`
    })
    export class AppComponent {
      @ViewChild('sidebar') sidebar?: SidebarComponent;
      public open(args: any) {
        console.log("Sidebar Opened");
      }
      public close(args: any) {
        console.log("Sidebar Closed");
      }
      openClick() {
        this.sidebar?.show();
      }
      toggleClick() {
        this.sidebar?.toggle();
      }
      closeClick() {
        this.sidebar?.hide();
      }
    }
  }

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';

```

```
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

Top and bottom sidebar in Angular Sidebar component

You can initialize Sidebar at the left and right positions by using the [position](#) property. It will automatically adjust the width of the main content.

You can also initialize Sidebar at the top and bottom positions in application level. For initializing Sidebar, you need to manually adjust the height of the main content.

In the following sample, the [toggle](#) method has been used to show or hide the top and bottom sidebar on button click.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="top-sidebar" #topSidebar [type]="type"
(open)="top_sidebar_open()" (close)="top_sidebar_close()" ">
    <div class="title">
      <div style="display:inline-block"> Top Sidebar
</div>
    </div>
    <div class="content">
      Place your top sidebar primary content here.
    </div>
  </ejs-sidebar>
  <ejs-sidebar id="bottom-sidebar" #bottomSidebar
[type]="type" (open)="bottom_sidebar_open()"
(close)="bottom_sidebar_close()" ">
    <div class="title">
      <div style="display:inline-block"> Bottom Sidebar
</div>
    <div class="content">
      Place your bottom sidebar primary content here.
    </div>
  </ejs-sidebar>
  <div class="e-main-content">
    <div class="sub-content">
      <p>Place your main content here.....</p>
      <div id="button-align">
        <button ej-button id="top-btn" class="toggle e-
btn e-info" (click)="topBtnClick()">Toggle Top Sidebar</button>
      </div>
      <div id="button-align">
        <button ej-button id="bottom-btn" class="toggle
e-btn e-info" (click)="bottomBtnClick()">Toggle Bottom Sidebar</button>
```

```

        </div>
    </div>
</div>`
}))
export class AppComponent {
    @ViewChild('topSidebar', { static: true }) topSidebar?: SidebarComponent;
    @ViewChild('bottomSidebar', { static: true }) bottomSidebar?:
SidebarComponent;
    public type: string = 'Push';
    // only for sample browser use
    constructor() {
    }
    topBtnClick() {
        this.topSidebar?.toggle();
    }
    bottomBtnClick() {
        this.bottomSidebar?.toggle();
    }
    top_sidebar_open() {
        let element: Element = document.getElementsByClassName("e-content-
animation")[0];
        (<HTMLElement>element).style.height =
((<HTMLElement>element).offsetHeight - 75) + "px";
        element.classList.add("top_content_animation");
        // Remove the e-left class in sidebar
        (this.topSidebar as SidebarComponent).element.classList.remove("e-
left");
        // Add the custom class to sidebar
        (this.topSidebar as
SidebarComponent).element.classList.add("top_sidebar");
    }
    top_sidebar_close() {
        let element: Element = document.getElementsByClassName("e-content-
animation")[0];
        (<HTMLElement>element).style.height =
((<HTMLElement>element).offsetHeight + 75) + "px";
        element.classList.remove("top_content_animation");
    }
    bottom_sidebar_open() {
        let element: Element = document.getElementsByClassName("e-content-
animation")[0];
        (<HTMLElement>element).style.height =
((<HTMLElement>element).offsetHeight - 75) + "px";
        element.classList.add("bottom_animation_content");
        // Remove the e-left class in sidebar
        (this.bottomSidebar as SidebarComponent).element.classList.remove("e-
left");
        // Add the custom class to sidebar
        (this.bottomSidebar as
SidebarComponent).element.classList.add("bottom_sidebar");
    }
    bottom_sidebar_close() {
        let element: Element = document.getElementsByClassName("e-content-
animation")[0];
        (<HTMLElement>element).style.height =
((<HTMLElement>element).offsetHeight + 75) + "px";
        element.classList.remove("bottom_animation_content");
    }
}

```

```
}
}
```

MAIN.TS

```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

APP.COMPONENT.CSS

```
/* sample-level styles */
body {
    overflow: hidden;
}
.e-main-content {
    text-align: center;
    background: #f5f5f5;
    font-size: 16px;
    overflow: auto;
}
.sub-content {
    height: 378px;
    padding: 50px;
}
#button-align {
    padding: 10px;
}
.title {
    text-align: center;
    font-size: 20px;
    padding: 5px;
}
.content {
    padding: 10px;
    text-align: center;
}
.toggle {
    width: 200px;
}
#top-sidebar,
#bottom-sidebar {
    background-color: rgb(25, 118, 210);
    color: #ffffff;
    overflow: hidden;
}
/* top sidebar element styles*/
.top_sidebar.e-open {
    transform: translateY(0%) !important;
    transition: transform 0.5s ease;
    visibility: visible;
}
.top_sidebar,
#top-sidebar.e-left,
.bottom_sidebar {
```

```

        width: 100% !important;
        float: left;
        height: 75px;
    }
    .top_sidebar,
    #top-sidebar.e-left {
        transform: translateY(-100%) !important;
    }
    .top_sidebar.e-close {
        transform: translateY(-100%);
    }
    .top_content_animation {
        margin-left: 0px !important;
        margin-top: 75px;
    }
}
/* end of top sidebar element styles*/
/* bottom sidebar element styles*/
.bottom_sidebar.e-open {
    transform: translateY(0%) !important;
    transition: transform 0.5s ease;
    visibility: visible;
    bottom: 0;
    top: unset !important;
    position: absolute;
}
.bottom_sidebar.e-close {
    transform: translateY(100%);
}
#bottom-sidebar.e-left {
    transform: translateY(+100%) !important;
}
.bottom_animation_content {
    margin-left: 0px !important;
}
}
/* end of bottom sidebar styles*/

```

Multiple sidebar in Angular Sidebar component

Two Sidebars can be initialized in a web page with same main content. Sidebars can be initialized on right side or left side of the main content using [position](#) property.

The HTML element with class name `e-main-content` will be considered as the main content and both the Sidebars will behave as side content to this main content area of a web page.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],

```

```

    template: `
      <ejs-sidebar id="default" #leftSidebar [width]='width'
      [type]='type' [closeOnDocumentClick]='closeOnDocumentClick'>
        <div class="title"> Left Sidebar content</div>
      </ejs-sidebar>
      <ejs-sidebar id="default1" #rightSidebar [width]='width'
      [position]='position'
        [type]='type'
      [closeOnDocumentClick]='closeOnDocumentClick'>
        <div class="title">Right Sidebar content</div>
      </ejs-sidebar>
      <div class="e-main-content" style="font-size: 16px;
padding: 100px 0; transform: translateX(0px); margin-left: 100px; margin-
right: 100px;">
        <p>Place your main content here.....</p>
        <div id="button-align">
          <button ejs-button id="toggle" class="e-btn e-
info" (click)="leftToggle()">Toggle Sidebar 1</button>
        </div>
        <div id="button-align">
          <button ejs-button id="toggle2" class="e-btn e-
info" (click)="rightToggle()">Toggle Sidebar 2</button>
        </div>
      </div>`
  })
  export class AppComponent {
    @ViewChild('leftSidebar') leftsidebar?: SidebarComponent;
    @ViewChild('rightSidebar') rightsidebar?: SidebarComponent;
    public width:string='250px';
    public position:string='Right';
    public type:string='Push';
    closeOnDocumentClick: any;
    leftToggle() {
      this.leftsidebar?.toggle();
    }
    rightToggle() {
      this.rightsidebar?.toggle();
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Sidebar with treeview in Angular Sidebar component

The following example demonstrates how to render TreeView component inside the Sidebar with dock state and how to achieve expand and collapse the functionalities simultaneously in the sidebar and Treeview.

On collapse, the LI elements of TreeView show icons only to represent the short sign of the hidden text content. On expand, hidden text content will be set to be visible.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule, TreeViewModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent, TreeViewComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, TreeViewModule],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <div id="wrapper">
    <div class="col-lg-12 col-sm-12 col-md-12"
id="sidebar-section">
      <div class="col-lg-12 col-sm-12 col-md-12">
        <!--header-section declaration -->
        <div class="main-header" id="header-section">
          <ul class="header-list">
            <li class="float-left header-style
icon-menu" id="hamburger" (click)="onClick()"></li>
            <li class="float-left header-style
nav-pane"><b>Navigation Pane</b></li>
          </ul>
        </div>
        <!--end of header-section declaration -->
        <!-- sidebar element declaration -->
        <ejs-sidebar id="sidebar-treeview"
class="dock-menu" #sidebarTreeViewInstance [enableDock]='enableDock'
[width]='width' [dockSize]='dockSize' (created)="onCreated($event)"
style="visibility: hidden"
[mediaQuery]='mediaQuery'
[target]='target' (close)="onClose($event)">
          <div class="main-menu">
            <div>
              <!-- Treevie element declaration
-->
              <ejs-treeview id="main-treeview"
#treeviewInstance [fields]='field' expandOn='Click'></ejs-treeview>
            </div>
          </div>
        </ejs-sidebar>
        <!-- end of sidebar element -->
        <!-- main-content declaration -->
        <div class="main-content" id="main-text">
          <div class="sidebar-content">
            <h2 class="sidebar-heading">
Responsive Sidebar With Treeview</h2>
            <p class="paragraph-content"> This is
a graphical aid for visualising and categorising the
site,
in the style of an expandable and
collapsible 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 class="line"></div>
                                <h2 class="sidebar-heading">Lorem
Ipsum Dolor</h2>
                                <p class="paragraph-content">Lorem
ipsum dolor sit amet, consectetur adipisicing elit, sed do
                                eiusmod tempor incididunt ut
labore et dolore magna aliqua. Ut enim ad minim veniam, quis
                                nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute
                                irure
                                dolor in reprehenderit in
voluptate velit esse cillum dolore eu fugiat nulla pariatur.</p>
                                <div class="line"></div>
                                <h2 class="sidebar-heading"> Lorem
Ipsum Dolor</h2>
                                <p class="paragraph-content">Lorem
ipsum dolor sit amet, consectetur adipisicing elit, sed do
                                eiusmod
                                tempor incididunt ut labore et
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
                                exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor
                                in
                                reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur
                                sint
                                occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id est
                                laborum.</p>
                                </div>
                                </div>
                                <!-- end of main-content -->
                                </div>
                                </div>
                                </div>`
}))
export class AppComponent {
    @ViewChild('sidebarTreeviewInstance')
    public sidebarTreeviewInstance?: SidebarComponent;
    @ViewChild('treeviewInstance')
    public treeviewInstance?: TreeViewComponent;
    public width: string = '290px';
    public enableDock: boolean = true;
    public dockSize:string ="44px";
    public mediaQuery: string = ('(min-width: 600px) ');
    public target: string = '.main-content';

```

```

    public data: Object[] = [
        {
            nodeId: '01', nodeText: 'Installation', iconCss: 'icon-microchip
icon',
        },
        {
            nodeId: '02', nodeText: 'Deployment', iconCss: 'icon-thumbs-up-
alt icon',
        },
        {
            nodeId: '03', nodeText: 'Quick Start', iconCss: 'icon-docs icon',
        },
        {
            nodeId: '04', nodeText: 'Components', iconCss: 'icon-th icon',
            nodeChild: [
                { nodeId: '04-01', nodeText: 'Calendar', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '04-02', nodeText: 'DatePicker', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '04-03', nodeText: 'DateTimePicker', iconCss:
'icon-circle-thin icon' },
                { nodeId: '04-04', nodeText: 'DateRangePicker', iconCss:
'icon-circle-thin icon' },
                { nodeId: '04-05', nodeText: 'TimePicker', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '04-06', nodeText: 'SideBar', iconCss: 'icon-
circle-thin icon' }
            ]
        },
        {
            nodeId: '05', nodeText: 'API Reference', iconCss: 'icon-code
icon',
            nodeChild: [
                { nodeId: '05-01', nodeText: 'Calendar', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '05-02', nodeText: 'DatePicker', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '05-03', nodeText: 'DateTimePicker', iconCss:
'icon-circle-thin icon' },
                { nodeId: '05-04', nodeText: 'DateRangePicker', iconCss:
'icon-circle-thin icon' },
                { nodeId: '05-05', nodeText: 'TimePicker', iconCss: 'icon-
circle-thin icon' },
                { nodeId: '05-06', nodeText: 'SideBar', iconCss: 'icon-
circle-thin icon' }
            ]
        },
        {
            nodeId: '06', nodeText: 'Browser Compatibility', iconCss: 'icon-
chrome icon'
        },
        {
            nodeId: '07', nodeText: 'Upgrade Packages', iconCss: 'icon-up-
hand icon'
        },
    ]

```

```

        nodeId: '08', nodeText: 'Release Notes', iconCss: 'icon-bookmark-
empty icon'
    },
    {
        nodeId: '09', nodeText: 'FAQ', iconCss: 'icon-help-circled icon'
    },
    {
        nodeId: '10', nodeText: 'License', iconCss: 'icon-doc-text icon'
    }
];
    public field:Object = { dataSource: this.data, id: 'nodeId', text:
'nodeText', child: 'nodeChild', iconCss: 'iconCss' };
    public onCreate(args: any) {
        (this.sidebarTreeviewInstance as
SidebarComponent).element.style.visibility = '';
    }
    public onClose(args: any) {
        (this.treeviewInstance as TreeViewComponent).collapseAll();
    }
    openClick() {
        if((this.sidebarTreeviewInstance as SidebarComponent).isOpen)
        {
            (this.sidebarTreeviewInstance as SidebarComponent).hide();
            (this.treeviewInstance as TreeViewComponent).collapseAll();
        }
        else {
            (this.sidebarTreeviewInstance as SidebarComponent).show();
            (this.treeviewInstance as TreeViewComponent).expandAll();
        }
    }
};

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Fixed sidebar in Angular Sidebar component

The Sidebar does not require any specific style to make it as a fixed one. By default, the Sidebar position will be in fixed state. The following example demonstrates that the Sidebar is rendered with a fixed position. The position of the Sidebar will not change when scrolling the main content area.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
    imports: [SidebarModule, ],
    standalone: true,
    selector: 'app-root',

```

```

styleUrls: ['./app.component.css'],
template: ` <div id='container'>
    <!-- declaration of sidebar element -->
    <div id="wrapper">
        <!--Sidebar element declaration -->
        <ejs-sidebar id="default-sidebar" #sidebar
[width]='width' (created)="onCreated($event)" style="visibility: hidden" >
            <div class="sidebar-header header-cover"
style="background-color:#0378d5">
                <div class="image-container">
                    <div class="sidebar-image">
                    </div>
                </div>
                <div style="padding: 0 0 5px 0;">
                    <a class="sidebar-brand"
href="#settings-dropdown">
                        john.doe@gmail.com
                    </a>
                    <span class="sf-icon-down
icon"></span>
                </div>
            </div>
            <!-- Sidebar navigation -->
            <ul class="nav sidebar-nav">
                <li>
                    <a href="#">
                        <i class="sf-icon-sidebar
sf-icon-file"></i>
                        <span class="e-text">
Inbox</span>
                    </a>
                </li>
                <li>
                    <a href="#">
                        <i class="sf-icon-sidebar
sf-icon-starred"></i>
                        <span class="e-text">
Starred</span>
                    </a>
                </li>
                <li>
                    <a href="#">
                        <i class="sf-icon-sidebar
sf-icon-recent"></i>
                        <span class="e-
text">Snoozed</span>
                    </a>
                </li>
                <li>
                    <a href="#">
                        <i class="sf-icon-sidebar
sf-icon-important"></i>
                        <span class="e-
text">Important</span>
                    </a>
                </li>
            </ul>
        </div>
    </div>
`

```

```

                <a href="#">
                    <i class="sf-icon-sidebar
sf-icon-offline"></i>
                    <span class="e-text">
Sent</span>
                </a>
            </li>
            <li>
                <a href="#">
                    <i class="sf-icon-sidebar
sf-icon-backup"></i>
                    <span class="e-text">
Draft</span>
                </a>
            </li>
        </ul>
    </ejs-sidebar>
    <!-- main content declaration -->
    <div>
        <div class="content">
            <div id="left">
                <span id="hamburger" class="e-
icons menu default" (click)="openClick()"></span>
            </div>
            <div id="center">
                <span>Inbox</span>
            </div>
            <div id="right">
                <span class="sf-icon-
search"></span>
            </div>
        </div>
        <div>
            <div class="e-control e-listview e-
list-template e-touch">
                <ul class="e-list-parent e-ul ">
                    <li class="e-list-group-item
e-level-1" role="group" data-uid="group-list-item-Today"
                        aria-level="1">
                        <div class="e-text-
content" role="presentation"><span class style="width: 100%; margin-left: 2%;
margin-top: -2%;">Today</span></div>
                    </li>
                    <li class="e-list-item">
                        <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                            <span class="e-avatar
e-icon sf-icon-profile"></span>
                            <div>
                                <span class="e-
list-item-header e-name">Albert Lives</span>
                                <span class="received
e-list-content e-second-heading">Opening for Sales Manager</span>
                                <span class="e-list-
item-header sf-icon-star">

```

```

    <span class="e-
list-text">Hello Uta Morgan,</span></span>
    </div>
</li>
<li class="e-list-item">
    <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
        <span class="e-avatar
e-icon sf-icon-profile"></span>
        <div>
            <span class="e-
list-item-header e-name">
                Ila
                Russo</span>
            </div>
            <span class="received
e-list-content e-second-heading">Business dinner invitation
            </span>
            <span class="e-list-
item-header sf-icon-star">
                <span class="e-
list-text">Hello Jelani Moreno,</span></span>
            </div>
        </li>
        <li class="e-list-item">
            <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                <span class="e-avatar
e-icon sf-icon-profile"></span>
                <div>
                    <span class="e-
list-item-header e-name">
                        Garth
                        Owen</span>
                    </div>
                    <span class="received
e-list-content e-second-heading">Application for Job Title</span>
                    <span class="e-list-
item-header sf-icon-star">
                        <span class="e-
list-text">Hello Ila Russo,</span></span>
                    </div>
                </li>
                <li class="e-list-item">
                    <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                        <span class="e-avatar
e-icon sf-icon-profile"></span>
                        <div>
                            <span class="e-
list-item-header e-name">Ursula Patterson</span>
                            </div>
                            <span class="received
e-list-content e-second-heading">Hello Kerry Best,</span>
                            <span class="e-list-
item-header sf-icon-star">

```

```

                                <span class="e-
list-text">Programmer Position Application</span></span>
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                                <span class="e-avatar
e-icon sf-icon-profile"></span>
                                <div>
                                <span class="e-
list-item-header e-name">
                                Nichole
Rivas</span>
                                </div>
                                <span class="received
e-list-content e-second-heading">Annual Conference</span>
                                <span class="e-list-
item-header sf-icon-star">
                                <span class="e-
list-text">Hi Igor Mccoy,</span></span>
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                                <span class="e-avatar
e-icon sf-icon-profile"></span>
                                <div>
                                <span class="e-
list-item-header e-name">
                                Nichole
Rivas</span>
                                </div>
                                <span class="received
e-list-content e-second-heading">Annual Conference</span>
                                <span class="e-list-
item-header sf-icon-star">
                                <span class="e-
list-text">Hi Igor Mccoy,</span></span>
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
wrapper e-list-avatar e-list-multi-line">
                                <span class="e-avatar
e-icon sf-icon-profile"></span>
                                <div>
                                <span class="e-
list-item-header e-name">
                                Nichole
Rivas</span>
                                </div>
                                <span class="received
e-list-content e-second-heading">Annual Conference</span>
                                <span class="e-list-
item-header sf-icon-star">

```

```

list-text">Hi Igor Mccoy,</span></span>
                                <span class="e-
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
                                <span class="e-avatar
                                <div>
                                <span class="e-
                                Nichole
                                </div>
                                <span class="received
                                <span class="e-list-
                                <span class="e-
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
                                <span class="e-avatar
                                <div>
                                <span class="e-
                                Nichole
                                </div>
                                <span class="received
                                <span class="e-list-
                                <span class="e-
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
                                <span class="e-avatar
                                <div>
                                <span class="e-
                                Ursula Patterson</span>
                                </div>
                                <span class="received
                                <span class="e-list-
                                <span class="e-
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
                                <span class="e-avatar
                                <div>
                                <span class="e-
                                Hello Kerry Best,</span>
                                <span class="e-list-
                                <span class="e-
                                </div>
                                </li>
                                <li class="e-list-item">
                                <div class="e-list-
                                <span class="e-avatar
                                <div>
                                <span class="e-
                                Programmer Position Application</span></span>

```



```

        </div>
      </li>
    </ul>
  </div>
</div>
<!--end of main content declaration -->
</div>
</div>`
  })
  export class AppComponent {
    @ViewChild('sidebar')
    public sidebar?: SidebarComponent;
    public width: string = '290px';
    public onCreated(args: any) {
      (this.sidebar as SidebarComponent).element.style.visibility = '';
    }
    openClick(): void {
      this.sidebar?.toggle();
    }
  }
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Sidebar with variation animation in Angular Sidebar component

In the following example, the sidebar is rendered with custom animation effects. Click the buttons available in the main content area to check how the custom animations works with sidebar.

Sidebar will automatically adjust expanding animation to match any custom size specified in **CSS** styles.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [SidebarModule, ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  template: ` <ejs-sidebar id="sidebar-element" class="sidebar" #sidebar
[showBackdrop]="showBackdrop" [closeOnDocumentClick]="closeOnDocumentClick"
(created)="onCreated($event)" style="visibility: hidden">
    <div class="title"> Sidebar content</div>
    <div class="sub-title">
      * Sidebar is rendered with animation effect
    </div>
    <div class="center-align">

```

```

        <button ejs-button id="close_btn"
(click)="closeClick_btn()" class="e-btn close-btn">Close Sidebar</button>
    </div>
</ejs-sidebar>
<div>
    <div class="title">Sidebar Transitions</div>
    <div class="sub-title"> * Click the below button to
render the Sidebar with animation effect.</div>
    <div style="padding:20px" class="center-align">
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="zoom()">Zoom Sidebar</button>
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="open_door()">Open Door</button>
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="bottom_top()">Bottom to Top</button>
    </div>
    <div style="padding:20px" class="center-align">
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="rotate()">Rotate Sidebar</button>
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="rotate_3d()">Rotate 3D</button>
        <button ejs-button id="toggle" class="e-btn e-info"
(click)="reverse()">Reverse Slide Out</button>
    </div>
</div>`
    ))
export class AppComponent {
    @ViewChild('sidebar') sidebar?: SidebarComponent;
    public showBackdrop: boolean = true;
    public closeOnClick: boolean = true;
    public onCreate(args: any) {
        (this.sidebar as SidebarComponent).element.style.visibility = '';
    }
    // Close Button
    closeClick_btn(): void {
        (this.sidebar as SidebarComponent).element.classList.remove("sidebar");
        (this.sidebar as SidebarComponent).element.classList.remove("rotate");
        (this.sidebar as SidebarComponent).element.classList.remove("w3-animate-
zoom");
        (this.sidebar as SidebarComponent).element.classList.remove("w3-animate-
bottom");
        (this.sidebar as SidebarComponent).element.classList.remove("rotate_3d");
        (this.sidebar as
SidebarComponent).element.classList.remove("reverse_slide_out");
        this.sidebar?.hide();
    };
    // Zoom Effect
    zoom():void{
        this.sidebar?.show();
        (this.sidebar as SidebarComponent).element.classList.add("w3-animate-
zoom");
    };
    // Open Door
    open_door():void{
        this.sidebar?.show();
        let sidebar_element: Element =document.getElementsByClassName("e-sidebar-
overlay")[0];

```

```

        sidebar_element.classList.add("move");
        (document.getElementsByClassName("move")[0] as any).style.transform
        ="rotateX(-20deg)";
    };
    // Bottom To Top
    bottom_top():void{
        this.sidebar?.show();
        (this.sidebar as SidebarComponent).element.classList.add("w3-animate-
bottom");
    };
    // Rotate
    rotate():void{
        this.sidebar?.show();
        (this.sidebar as SidebarComponent).element.classList.add("rotate");
    };
    // Rotate 3D
    rotate_3d():void{
        this.sidebar?.show();
        (this.sidebar as SidebarComponent).element.classList.add("rotate_3d");
    };
    // Reverse Slide Out
    reverse():void{
        this.sidebar?.show();
        (this.sidebar as
SidebarComponent).element.classList.add("reverse_slide_out");
    };
}

```

MAIN.TS

```

import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

Hide sidebar in Angular Sidebar component

The following example demonstrates how to hide master page sidebar. Initially sidebar is rendered with master page. While navigate to another page, it hides the master page sidebar using angular routing.

Refer the Sidebar component in `app.component.html`

```

`html
<router-outlet>
<ul>
<li>
<a routerLink="/">Master page</a>
</li>
<li>
<a routerLink="/firstPage">First page</a>

```

```
</li>
<li>
<a routerLink="/secondPage">Second page</a>
</li>
</ul>
</router-outlet>
<ejs-sidebar id="parent-sidebar" #sidebar width="260px" type="Push" class="e-hidden"
(created)="onCreated()">
<div class="title">
<span id="close" class="e-icons" (click)="closeClick()"></span>
</div>
<div class="sub-title"> Application Page Sidebar</div>
</ejs-sidebar>
<div id="myCarousel" class="carousel slide" data-ride="carousel" data-interval="6000">
<h2 class="sidebar-heading"> Responsive Sidebar With Treeview</h2>
<p class="paragraph-content">
This is a graphical aid for visualising and categorising the site, in the style of an expandable and
collapsible 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 class="line"></div>
<h2 class="sidebar-heading">Lorem Ipsum Dolor</h2>
<p class="paragraph-content">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
</p>
```

```
<div class="line"></div>
<h2 class="sidebar-heading"> Lorem Ipsum Dolor</h2>
<p class="paragraph-content">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="line"></div>
<h2 class="sidebar-heading"> Lorem Ipsum Dolor</h2>
<p class="paragraph-content">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="line"></div>
<h2 class="sidebar-heading"> Lorem Ipsum Dolor</h2>
<p class="paragraph-content">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="line"></div>
<h2 class="sidebar-heading"> Lorem Ipsum Dolor</h2>
<p class="paragraph-content">
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
```

reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

</p>

</div>

,

Add this below code in `app.component.ts`

``typescript`

```
import { Component, ViewChild, AfterViewInit, ViewEncapsulation } from '@angular/core';
```

```
// importing Sidebar components from the ej2-angular-navigations package
```

```
import { SidebarComponent } from '@syncfusion/ej2-angular-navigations';
```

```
import { Router, NavigationEnd } from '@angular/router';
```

```
@Component({
```

```
  selector: 'app-root',
```

```
  templateUrl: './app.component.html',
```

```
  styleUrls: ['./app.component.css'],
```

```
  encapsulation: ViewEncapsulation.None
```

```
})
```

```
export class AppComponent implements AfterViewInit { {
```

```
  @ViewChild('sidebar')
```

```
  // Instance of the Sidebar in the main page
```

```
  public sidebarInstance: SidebarComponent;
```

```
  // The below variable will hold the URL of the navigation page when router event is triggered
```

```
  public urlValue: String;
```

```
  constructor(router: Router) {
```

```
    // registering router module in the component
```

```
    router.events.forEach((event) => {
```

```
      if (event instanceof NavigationEnd) {
```

```
        this.urlValue = event.url;
```

```
        //Based on the routed URL, Sidebar in the main page will expand or collapse.
```

```
        this.checkURL();
```

```
      }
```

```
    });
```

```
  }
```

```
onCreated() {  
  // This event will trigger whenever a Sidebar in the main page is rendered.  
  if (this.sidebarInstance.element.classList.contains("e-hidden")) {  
    this.sidebarInstance.element.classList.remove("e-hidden");  
    this.checkURL();  
  }  
}  
  
checkURL() {  
  //Based on the routed URL, Sidebar in the main page will expand or collapse.  
  (!this.urlValue || this.urlValue !== "/" ) ? this.sidebarInstance.hide() : this.sidebarInstance.show();  
}  
  
closeClick() {  
  //On clicking the close icon, Sidebar will get collapsed  
  this.sidebarInstance.hide();  
}  
  
ngAfterViewInit() {  
  if (!this.urlValue || this.urlValue !== "/" ) {  
    this.sidebarInstance.hide();  
  }  
}  
}
```

The following Sample demonstrates how to Hide the sidebar using angular service in routing application. Refer to this sample.

[Sample](#)