

USER GUIDE

# Essential Studio

## for EJ2 Angular

---

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

Gantt .....	23
Getting started with Angular Gantt component.....	23
Setup Angular Environment.....	23
Create an Angular Application .....	23
Adding Syncfusion Gantt package.....	23
Registering Gantt Module.....	23
Adding CSS reference.....	24
Add Gantt component .....	24
Binding Gantt with data .....	25
Module injection.....	26
Mapping task fields.....	27
Defining timeline.....	28
Enabling toolbar .....	30
Enabling editing.....	31
Enabling predecessors or task relationships.....	39
Assigning resources.....	40
Enable filtering .....	42
Enable sorting .....	44
Defining eventmarkers.....	46
Run the application .....	47
Module in Angular Gantt component.....	49
Data binding in Angular Gantt component.....	49
Local data .....	50
Remote data.....	52
Split task.....	71
Improve performance by disabling validations .....	75
Limitations.....	81
Observable in Angular Gantt component .....	81
Observable binding using Async pipe .....	81
Data binding.....	81
Columns in Angular Gantt component .....	84
Column types .....	84
Column width.....	86
Column formatting.....	88
Align the text of content .....	96

Render boolean value as checkbox.....	98
AutoFit columns .....	100
Locked columns.....	102
Show or hide columns.....	104
Controlling Gantt actions .....	108
Customize column styles.....	110
Manipulating columns .....	110
Responsive columns.....	114
Timeline in Angular Gantt component .....	116
Timeline view modes .....	116
Week start day customization .....	123
Customize automatic timescale update action.....	125
Timeline cells tooltip .....	126
Splitter in Angular Gantt component.....	127
Splitter.....	127
Change splitter position dynamically.....	129
Task scheduling in Angular Gantt component.....	130
Automatically scheduled Tasks .....	130
Manually scheduled Tasks .....	132
Custom .....	134
Unscheduled Tasks.....	137
Define unscheduled tasks in data source .....	138
Working time range .....	140
Weekend/non-working days.....	141
Duration units .....	142
Taskbar in Angular Gantt component.....	145
Taskbar customization .....	145
Multi Taskbar support in project view .....	150
Connector lines .....	151
Enable tooltip.....	152
Tooltip template .....	155
Tooltip Touch interaction.....	158
Critical path in Angular Gantt component.....	158
Customize taskbar in critical path.....	160
Undo Redo in Angular Gantt component .....	161

Configure the feature set for undo redo actions .....	161
Configuring the Storage Step Count for Undo and Redo Actions .....	163
Perform undo redo actions programatically .....	165
Retrieve undo and redo stack collection .....	166
Clear undo and redo collection .....	168
Toolbar in Angular Gantt component .....	170
Built-in toolbar items .....	170
Custom toolbar items .....	171
Built-in and custom items in toolbar .....	173
Enable/disable toolbar items .....	174
Add input elements in toolbar .....	176
Managing tasks in Angular Gantt component .....	178
Troubleshoot: Editing works only when primary key column is defined .....	179
Open new task dialog with default values .....	179
Cell edit type and its params .....	181
Taskbar editing tooltip .....	184
Scrolling in Angular Gantt component .....	185
Set width and height .....	185
Responsive with the parent container .....	186
Virtual scroll in Angular Gantt component .....	187
Row virtualization .....	187
Timeline virtualization .....	192
Limitations for virtual scroll .....	198
Resources in Angular Gantt component .....	198
Resource collection .....	198
Assign resource .....	199
Add/Edit resource collection .....	202
Resource view in Angular Gantt component .....	203
Resource task .....	203
Resource overAllocation .....	207
Unassigned task .....	210
Enable taskbar drag and drop .....	211
Filtering in Angular Gantt component .....	214
Filter hierarchy modes .....	214
Initial filter .....	216



Filter operators .....	218
Diacritics.....	218
Filtering a specific column by method .....	220
Clear filtered columns.....	222
Sorting in Angular Gantt component.....	224
Sorting column on Gantt initialization .....	226
Sorting column dynamically.....	228
Clear all the sorting dynamically.....	229
Sorting events .....	231
Sorting Custom Columns.....	233
Touch interaction .....	235
Selection in Angular Gantt component.....	236
Selection mode .....	236
Toggle selection .....	238
Clear selection.....	240
Get selected row indexes and records.....	241
Multiple Selection based on condition .....	243
Touch interaction.....	245
See Also .....	245
Rows in Angular Gantt component.....	245
Row height .....	245
Expand/Collapse Row .....	247
Customize rows and cells.....	251
Customize rows.....	253
Styling alternate rows .....	254
Row spanning.....	256
Clip mode .....	257
Export.....	259
Excel export in Angular Gantt component.....	259
Export.....	260
Context menu in Angular Gantt component .....	281
Custom context menu items.....	283
Touch interaction.....	285
State persistence in Angular Gantt component.....	285
Get or set localStorage value .....	285

Prevent columns from persisting .....	285
Persist the header template and header Text .....	288
Timezone in Angular Gantt component.....	290
Understanding date manipulation in JavaScript.....	290
Display same time everywhere with no time difference .....	290
CRUD operations with timezone.....	292
Timezone methods .....	295
Global local in Angular Gantt component .....	296
Localization .....	296
Internationalization.....	300
Right to left (RTL) .....	302
See Also .....	305
Accessibility in Angular Gantt component.....	305
WAI-ARIA attributes.....	306
Keyboard navigation .....	307
Ensuring accessibility .....	308
See also .....	308
Ej1 api migration in Angular Gantt component.....	308
Data Binding and Task mapping.....	308
Sorting .....	309
Filtering .....	310
Searching.....	311
Selection.....	311
Editing .....	312
Columns .....	315
Toolbar .....	316
ToolTip .....	317
Timeline.....	318
Rows.....	319
Resources .....	321
Baseline .....	321
Context Menu .....	322
Scheduling Tasks .....	322
Appearance and Customizations .....	323
Stripline .....	325

Holidays.....	325
Others .....	325
Style and appearance in Angular Gantt component.....	328
Grid lines .....	329
How To .....	331
Open add edit dialog in Angular Gantt component.....	331
SetScrollTop in Angular Gantt component .....	332
Change schedule dates in Angular Gantt component .....	333
Copy paste records in Angular Gantt component .....	335
Maintain record index in Angular Gantt component.....	337
Custom field in Angular Gantt component .....	340
Maintain zoom to fit in Angular Gantt component .....	343
Drag and drop from another component in Angular Gantt component .....	346
Grid.....	349
Getting started with Angular Grid component .....	349
Setup Angular Environment.....	349
Create an Angular Application .....	349
Adding Syncfusion Grid package.....	350
Registering Grid Module .....	350
Adding CSS reference.....	351
Add Grid component.....	351
Defining Row Data .....	351
Defining Columns .....	354
Module Injection.....	356
Enable Paging.....	356
Enable Sorting.....	357
Enable Filtering .....	358
Enable Grouping.....	359
Run the application .....	360
See Also .....	361
Module in Angular Grid component .....	362
Data binding in Angular Grid component .....	363
Loading indicator .....	363
Refresh the datasource using property .....	365
Dynamically change the datasource or columns or both .....	367

See also .....	371
Connecting to Adaptors .....	371
UrlAdaptor in Syncfusion Angular Grid Component .....	371
ODataV4Adaptor in Syncfusion Angular Grid Component .....	394
WebApiAdaptor in Syncfusion Angular Grid Component .....	411
Connecting GraphQL Service with Angular Grid Component .....	428
Grid aggregates type .....	432
Represents parameters for querying data, including sorting, filtering, etc.....	432
Represents an order type .....	433
Represents the result of a query, including the data and count .....	433
Represents a query to fetch orders with specified data manager parameters .....	433
WebMethodAdaptor in Syncfusion Angular Grid Component .....	454
Adaptive in Angular Grid component .....	480
Render adaptive dialogs.....	480
Vertical row rendering .....	482
See Also .....	487
Performance tips for Angular DataGrid Component .....	487
How to improve loading performance by binding large dataset.....	487
How to improve loading performance by binding large data by showing custom text or element.	488
How to improve loading performance by referring individual script and CSS .....	489
How to update cell values without frequent server calls .....	489
How to optimize server-side data operations with adaptors .....	489
How to avoid MaxJsonLength error while passing large amount of records .....	489
Optimizing Angular app performance with multiple grids and templates .....	490
Microsoft excel limitation while exporting millions of records to excel file format.....	491
Columns in Angular Grid Component .....	491
Column types .....	491
Column Width .....	493
Column formatting.....	495
Align the text of content .....	501
Render boolean value as checkbox.....	502
How to prevent checkbox in the blank row .....	503
AutoFit columns .....	505
Locked columns.....	509
Show or hide columns.....	511

Controlling Grid actions .....	515
Customize column styles.....	516
Manipulating columns .....	516
Responsive columns.....	520
See also .....	521
Row in Angular Grid component.....	522
Customize row styles .....	522
Row height .....	527
Row hover .....	530
Row pinning (Frozen).....	533
Adding a new row programmatically.....	538
Show or hide a row using an external actions .....	540
How to get the row data and element.....	542
See Also .....	543
Cell in Angular Grid component.....	544
Displaying the HTML content.....	544
Autowrap the grid content .....	546
Customize cell styles .....	548
Clip Mode .....	553
Tooltip .....	555
Grid lines .....	560
See Also .....	561
Edit in angular grid component .....	561
Toolbar with edit option .....	563
Disable editing for particular column .....	565
Editing template column.....	567
Customize delete confirmation dialog.....	569
Update boolean column value with a single click.....	571
Edit enum column .....	573
Edit complex column.....	575
Edit foreign key column .....	577
How to perform CRUD action externally .....	579
Troubleshoot editing works only for first row .....	584
How to make a grid column always editable .....	584
See also .....	586

Sorting in Angular Grid component .....	586
Initial sorting .....	587
Multi-column sorting .....	589
Prevent sorting for particular column.....	590
Sort order .....	590
Custom sorting .....	591
Touch interaction .....	592
Sort foreign key column based on text .....	593
Perform sorting based on its culture .....	596
How to customize sort icon .....	598
Sort columns externally .....	599
Sorting Events .....	605
See Also .....	606
Grouping in Angular Grid component.....	606
Initial group .....	607
Prevent grouping for particular column .....	608
Hide drop area .....	609
Show the grouped column.....	611
Reordering on grouped columns .....	612
Sort grouped columns in descending order during initial grouping .....	613
Group with paging.....	614
Group by format .....	615
Show grouped rows based on page size .....	616
Collapse all grouped rows at initial rendering .....	617
Group or ungroup column externally .....	619
Expand or collapse externally .....	621
Clear grouping.....	624
Grouping Events.....	626
Limitations.....	627
See also .....	627
Filtering in Angular Grid component .....	628
Initial filter .....	629
Filter operators .....	633
Diacritics filter .....	635
Filtering with case sensitivity .....	636

Enable different filter for a column .....	638
Change default filter operator for particular column .....	640
Filter grid programmatically with single and multiple values using method.....	641
How to get filtered records.....	643
Clear filtering using methods.....	646
Filtering events.....	647
See Also .....	648
Scrolling in Angular Grid component .....	649
Set width and height .....	649
Responsive with parent container .....	650
Sticky header.....	651
Scroll to selected row.....	652
Hide the empty placeholder of scrollbar .....	654
Searching in Angular Grid component .....	655
Initial search .....	656
Search by external button.....	658
Search specific columns .....	660
Search on each key stroke .....	661
Perform search based on column formatting.....	662
Perform search operation in Grid using multiple keywords.....	664
How to ignore accent while searching.....	667
Highlight the search text.....	669
Clear search by external button.....	671
See also .....	672
Paging in Angular Grid component .....	672
Customize the pager options .....	673
Pager template.....	679
Pager with page size dropdown.....	681
How to navigate to particular page .....	683
How to get the pager element.....	685
Dynamically calculate page size based on element height.....	685
Render pager at the top of the grid .....	687
Pager events.....	688
See Also .....	690
Selection in Angular Grid component.....	690

Selection mode .....	692
Touch interaction .....	693
Toggle selection .....	694
Clear all selection programmatically.....	696
Persist selection .....	698
See Also .....	699
Aggregates in Angular Grid component.....	699
Built-in aggregate types .....	701
Multiple aggregates for a column.....	703
See Also .....	704
Print in Angular Grid component.....	705
Page setup.....	706
Print by external button.....	706
Print visible Page.....	707
Print only selected records .....	708
Print the hierarchy grid .....	710
Print the master detail grid .....	711
Print large number of columns .....	713
Show or hide columns while printing .....	714
Limitations of printing large data.....	715
Retain grid styles while printing.....	716
Print grid along with other components.....	717
Hierarchy grid in Angular Grid component.....	719
Bind hierarchy grid with different field.....	721
Expand child grid initially .....	722
Dynamically load child grid data .....	724
Dynamically bind data to child grid based on parent row data.....	725
Adding record in child grid.....	727
Template column in child grid .....	729
How to get parent detail in child grid .....	730
Expand all by external button.....	732
Hide the expand/collapse icon in parent row when no record in child grid .....	734
Customize the child grid.....	736
See Also .....	770
State Management in Angular Grid component.....	770



Restore initial Grid state .....	772
Restore to specific state version .....	774
Restore to previous state.....	776
Maintaining custom query in a persistent state .....	778
Get or set local storage value .....	779
Prevent columns from persisting .....	780
Add to persist .....	781
Toolbar in Angular Grid component .....	784
Enable or disable toolbar items .....	786
Add toolbar at the bottom of grid .....	787
Customize toolbar buttons using CSS .....	789
See Also .....	790
Pdf export in Angular Grid component.....	790
Show spinner while exporting.....	792
Binding custom data source while exporting.....	793
Exporting with custom aggregate .....	794
Exporting with cell and row spanning.....	796
Exporting with custom date format.....	799
Exporting multiple grids .....	800
Exporting hierarchy grid.....	803
Remove header row while exporting.....	806
See also .....	807
Excel exporting in Angular Grid component .....	807
Show spinner while exporting.....	809
Binding custom data source while exporting.....	810
Exporting with custom aggregate .....	812
Exporting with cell and row spanning.....	813
Exporting with custom date format.....	816
Exporting multiple grids .....	817
Exporting hierarchy grid.....	821
Remove header row while exporting.....	824
How to add formula for the cell while exporting.....	826
Limitations.....	828
See Also .....	828
Global local in Angular Grid component.....	828

Localization .....	828
Internationalization.....	868
Right to Left - RTL.....	873
See Also .....	878
Accessibility in Angular Grid component .....	878
WAI-ARIA attributes.....	879
Keyboard interaction .....	880
Ensuring accessibility .....	886
See also .....	888
Clipboard in Angular Grid component .....	888
Copy to clipboard by external buttons .....	889
AutoFill .....	890
Paste.....	892
Context menu in Angular Grid component.....	894
Custom context menu items.....	897
Show context menu on left click.....	898
Enable or disable context menu items .....	900
Show or hide context menu items .....	902
Style and appearance in Angular Grid component.....	904
Customizing the grid root element .....	905
See Also .....	907
Ej1 api migration in Angular Grid component .....	907
Sorting.....	907
Grouping .....	908
Filtering .....	909
Searching.....	910
Paging.....	910
Selection.....	911
Editing .....	914
Resizing .....	918
Reordering .....	918
Context Menu .....	918
Toolbar .....	919
GridLines .....	920
Templates.....	920

Row/Column Drag and Drop .....	921
Frozen Rows and Columns .....	922
ForeignKey .....	922
Auto Wrap .....	922
Responsive .....	923
State Persistence.....	923
Right to Left - RTL.....	923
ToolTip .....	923
Aggregate/Summary .....	923
Grid Export .....	924
Columns .....	924
Row .....	926
Show/Hide Columns.....	927
Column Chooser.....	927
Header.....	927
DataSource.....	928
Print.....	928
Scrolling.....	928
PrimaryKey .....	928
Grid.....	929
How To .....	932
Enable disable grid and its actions in Angular Grid component .....	932
Customize the edit dialog in Angular Grid component.....	934
Example of angular ui grid to edit a cell using cascading drop down list in Angular Grid component .....	936
Hide sorting in excel filter in Angular Grid component .....	939
Exporting grid in cordova application in Angular Grid component .....	940
Customize pager drop down in Angular Grid component .....	942
Add params for filtering in Angular Grid component .....	943
Select grid rows based on certain condition in Angular Grid component.....	944
Get row cell index in Angular Grid component .....	945
Display null values at bottom in Angular Grid component.....	946
Enable editing in single click in Angular Grid component.....	948
Use custom helper inside the loop with templates in Angular Grid component .....	951
Resize the grid in various dimension in Angular Grid component.....	952

Ngx translate pipe for header text in Angular Grid component .....	954
Unit Jasmine testing in Angular Grid component .....	957
Customize the empty record template in Angular Grid component .....	967
HeatMap Chart.....	968
Getting started with Angular Heatmap chart component.....	968
Setup Angular Environment.....	968
Create an Angular Application .....	969
Installing Syncfusion Heatmap package.....	969
Registering Heatmap Module .....	970
Add Heatmap component.....	970
Module injection.....	971
Populate heat map with data .....	972
Enable axis labels .....	973
Add heat map title .....	974
Enable legend.....	975
Add data label.....	976
Add custom cell palette .....	978
Enable tooltip.....	979
Working with data in Angular Heatmap chart component .....	981
Data adaptor .....	981
Empty points .....	989
Binding nested JSON data .....	990
See Also .....	993
Bubble heatmap in Angular Heatmap chart component.....	993
Bubble types .....	993
Rendering mode in Angular Heatmap chart component .....	1008
Axis in Angular HeatMap chart component.....	1009
Types .....	1009
Inversed axis.....	1013
Opposed axis.....	1015
Axis labels customization .....	1016
Axis intervals .....	1024
Axis label increment.....	1026
Limiting labels in date-time axis.....	1027
Multilevel Labels .....	1030

Palette in Angular Heatmap chart component.....	1032
Palette types .....	1033
Defining color stops .....	1036
See Also .....	1037
Legend in Angular Heatmap chart component.....	1037
Legend types .....	1039
Placement .....	1040
Alignment.....	1041
Legend dimensions .....	1043
Paging for legend .....	1044
Smart Legend .....	1046
Legend Selection .....	1047
Legend Title .....	1049
Appearance in Angular Heatmap chart component.....	1050
Cell customization.....	1050
Background color .....	1054
Margin.....	1056
Title .....	1057
Data label .....	1058
See Also .....	1069
Dimensions in Angular Heatmap chart component.....	1069
Size for container .....	1069
Size for heat map .....	1069
In pixel.....	1069
In percentage .....	1071
Tooltip in Angular Heatmap chart component.....	1072
Default tooltip.....	1072
Tooltip template .....	1073
Customize the appearance of Tooltip.....	1074
Selection in Angular HeatMap chart component .....	1076
Enable single cell selection .....	1077
Accessibility in Angular HeatMap chart component .....	1080
WAI-ARIA attributes.....	1081
Screen reading in HeatMap .....	1081
Ensuring accessibility .....	1081

See also .....	1082
Events in Angular Heatmap chart component.....	1082
cellClick.....	1082
cellDoubleClick.....	1083
cellRender .....	1084
cellSelected .....	1086
created .....	1087
legendRender.....	1088
load .....	1090
loaded .....	1091
resized.....	1092
tooltipRender .....	1093
How to.....	1095
Tooltip template in Angular Heatmap chart component .....	1095
Legend customization in Angular Heatmap chart component .....	1096
Ej1 api migration in Angular Heatmap chart component .....	1098
Members.....	1098
Events.....	1101
Image Editor.....	1101
Getting started with Angular Image editor component .....	1101
Dependencies.....	1101
Setup Angular environment.....	1102
Create an Angular application .....	1102
Installing Syncfusion Image Editor package .....	1102
Adding Image Editor module .....	1102
Adding Syncfusion Image Editor component.....	1103
Adding CSS reference.....	1103
Running the application.....	1103
End-user capabilities in the Image Editor component.....	1104
Open an image .....	1104
Zooming .....	1105
Panning .....	1107
Cropping and image transformation.....	1107
Annotations.....	1108
Filtering and fine-tune .....	1109

Undo and redo the operations .....	1110
Reset an image .....	1111
Export an image .....	1112
Open and save in the Angular Image Editor component.....	1113
Open.....	1113
Save .....	1114
File opened event .....	1115
Saving event .....	1116
Created event.....	1116
Destroyed event.....	1116
Reset an image.....	1116
Selection cropping in the Angular Image Editor component.....	1116
Insert custom / square / circle region.....	1117
Insert selection based on aspect ratio .....	1118
Resize selections .....	1119
Crop an image .....	1119
Cropping event.....	1120
Annotation in the Angular Image Editor component .....	1120
Text annotation.....	1121
Freehand drawing .....	1127
Shape annotation.....	1130
Delete a shape .....	1133
Image annotation.....	1135
Transform in the Angular Image Editor component.....	1136
Rotate an image .....	1136
Flip an image .....	1137
Straightening in the Angular Image Editor control .....	1138
Zoom in or out an image.....	1140
Panning an image.....	1142
Zooming event .....	1143
Rotating event.....	1143
Flipping event.....	1143
Toolbar in the Angular Image Editor component .....	1144
Built-in toolbar items .....	1144
Add a custom toolbar items.....	1144

Show or hide a toolbar.....	1145
Show or hide a toolbar item .....	1146
Toolbar template .....	1147
Customize Contextual Toolbar.....	1148
Toolbar item clicked event.....	1150
Toolbar created event.....	1151
Add an additional contextual Toolbar item to text shape .....	1151
Quick access toolbar in the Angular Image Editor component .....	1152
Add a custom toolbar item .....	1152
Undo and redo actions in the Angular Image Editor .....	1153
Undo the action .....	1153
Redo the action.....	1154
Filters in the Angular Image Editor component.....	1155
Apply filter effect .....	1155
Image filtering event.....	1156
Finetune in Angular Image Editor component.....	1157
Adjust the brightness, contrast, or sharpness .....	1157
Adjust the hue, exposure, blur, or opacity .....	1158
Finetune value changing event.....	1160
Frames.....	1160
Apply frame to the Image .....	1160
Frame changing event.....	1162
Resize .....	1162
Apply resize to the image.....	1162
Resizing event .....	1164
Localization in the Angular Image Editor component .....	1164
Accessibility in Angular Image Editor component .....	1167
Keyboard interaction .....	1168
Ensuring accessibility .....	1168
See also .....	1168
In-place Editor.....	1168
Getting started with Angular Inplace editor component .....	1168
Getting Started with Angular CLI .....	1168
Installing Syncfusion In-place Editor package.....	1169
Adding In-place Editor module .....	1170



Adding In-place Editor component .....	1171
Adding CSS reference.....	1171
Running the application .....	1172
Add the In-place Editor with Textbox .....	1173
Configuring DropDownList.....	1174
Integrate DatePicker .....	1174
Two-way binding.....	1176
Submitting data to the server (save) .....	1177
Refresh with modified value .....	1177
See Also .....	1178
Controls in Angular Inplace editor component.....	1179
Model configuration .....	1182
Configuration in Angular Inplace editor component.....	1183
Rendering modes .....	1183
Event actions for editing .....	1186
Action on focus out .....	1187
Display modes .....	1188
See Also .....	1189
Buttons in Angular Inplace editor component .....	1189
See Also .....	1191
Server actions in Angular Inplace editor component .....	1191
Data binding in Angular Inplace editor component.....	1193
Local Data Binding.....	1193
Remote Data Binding .....	1194
Integration in Angular Inplace editor component .....	1196
As a string.....	1196
As a ng-template .....	1196
Validation in Angular Inplace editor component.....	1197
Validation Rules .....	1198
Style in Angular Inplace editor component .....	1198
Customizing the In-place Editor text.....	1198
Customizing the In-place Editor action buttons .....	1199
Localization in Angular Inplace editor component .....	1199
Localization .....	1199
Right to left .....	1201

Format.....	1202
Accessibility in Angular Inplace editor component .....	1203
Keyboard interaction .....	1204
Ensuring accessibility .....	1204
See also .....	1204
How To .....	1204
Dynamic edit mode in Angular Inplace editor component.....	1204
Disable edit mode in Angular Inplace editor component .....	1205
Custom indication in Angular Inplace editor component .....	1206
Custom animation in Angular Inplace editor component .....	1207

## Gantt

### Getting started with Angular Gantt component

This section explains you the steps required to create a simple Gantt component and demonstrate the basic usage of the Gantt 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
`
```

#### Adding Syncfusion Gantt package

All the available Essential JS 2 packages are published in [npmjs.com](#) registry.

To install Gantt component, use the following command.

```
`bash
ng add @syncfusion/ej2-angular-gantt --save
`
```

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

#### Registering Gantt Module

Import Gantt module into Angular application(**app.module.ts**) from the package **@syncfusion/ej2-angular-gantt** [**src/app/app.module.ts**].

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the GanttModule for the Gantt component
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { AppComponent } from './app.component';
@NgModule({
//declaration of ej2-angular-gantt module into NgModule
```

```
imports: [ BrowserModule, GanttModule ],
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-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-layouts/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-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-treegrid/styles/material.css';
@import '../node_modules/@syncfusion/ej2-gantt/styles/material.css';
```

### Add Gantt component

Modify the template in `[src/app/app.component.ts]` file to render the gantt component.

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

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

@Component({
  selector: 'app-root',
  // specifies the template string for the Gantt component
  template: <ejs-gantt> </ejs-gantt>
```

```
})  
export class AppComponent implements OnInit {  
  ngOnInit(): void {  
  }  
}  
`
```

### Binding Gantt with data

Bind data for the Gantt component by using [dataSource](#) property.

It accepts either array of JavaScript object or **DataManager** instance.

```
`typescript  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  template: <ejs-gantt [dataSource]='data'> </ejs-gantt>  
})  
export class AppComponent implements OnInit {  
  public data: Object[];  
  ngOnInit(): void {  
    this.data = [  
      {  
        TaskID: 1,  
        TaskName: 'Project Initiation',  
        StartDate: new Date('04/02/2019'),  
        EndDate: new Date('04/21/2019'),  
        subtasks: [  
          { TaskID: 2, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },  
        ],  
      },  
    ]  
  }  
}
```

```

TaskID: 5,
TaskName: 'Project Estimation',
StartDate: new Date('04/02/2019'),
EndDate: new Date('04/21/2019'),
subtasks: [
{ TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration:
3, Progress: 50 },
{ TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
{ TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress:
50 }
]
},
];
}
}
,

```

### Module injection

The Gantt component was segregated into individual feature-wise modules. To use its feature, you need to inject its feature service in the AppModule.

Find the relevant feature modules and descriptions as follows:

- [Edit](#) : Inject this module to use the editing feature.
- [Filter](#) : Inject this module to use the filtering feature.
- [Sort](#) : Inject this module to use the sorting feature.
- [Selection](#) : Inject this module to use the selection feature.
- [Toolbar](#) : Inject this module to use the toolbar items.
- [DayMarkers](#) : Inject this module to highlight the days.

Now, import the above-mentioned modules from the Gantt package and inject them into **provider** section of **AppModule** like following code:

```

`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { GanttComponent, EditService, FilterService, SortService, SelectionService,
ToolbarService, DayMarkersService } from '@syncfusion/ej2-angular-gantt';
@NgModule({
imports: [

```

```

BrowserModule,
],
declarations: [AppComponent, GanttComponent],
bootstrap: [AppComponent],
providers: [ EditService , FilterService, SortService, SelectionService,ToolbarService,DayMarkersService ]
})
`

```

### Mapping task fields

The data source fields that are required to render the tasks are mapped to the Gantt control using the [taskFields](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',

```

```

        startDate: new Date('04/02/2019'),
        endDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', startDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', startDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', startDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
}
}

```

### MAIN.TS

```

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

```

### Defining timeline

The Gantt has an option to define timeline using [timelineSettings](#) property with various options. Using this property we can customize the Gantt timeline.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,
    selector: 'app-root',
    template:
`<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
gantt>`,
    encapsulation: ViewEncapsulation.None
})

```



```

export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.timelineSettings = {
      topTier: {
        format: 'MMM dd, yyyy',
        unit: 'Week',
      },
      bottomTier: {
        unit: 'Day',
      },
    };
  }
}

```

**MAIN.TS**

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

## Enabling toolbar

The [toolbar](#) property is used to add the toolbar items like Add, Remove, Edit, Update, Delete, Expand All, Collapse All in Gantt.

To use toolbar, inject the **ToolbarService** in the provider section of **AppModule**.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
this.editSettings = {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
};
this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll'];
}
}

```

## MAIN.TS

```

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

```

## Enabling editing

The editing feature enables you to edit the tasks in Gantt component. It can be enabled by using the [editSettings.allowEditing](#) and [editSettings.allowTaskbarEditing](#) properties

To edit the tasks, inject the [EditService](#) in the provider section of `AppModule`.

The following editing options are available to update the tasks in Gantt:

- Cell
- Dialog
- Taskbar
- Connector line

### Cell editing

Modify the task details through cell editing by setting the `edit mode` property as `Auto`. To enable edit support `Edit` module should be injected in Gantt. If `Edit` module is not injected, you cannot do any editing action in Gantt.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [columns]="columns" [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public toolbar?: string[];
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      }
    ],
  },
}
```

```

        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.editSettings = {
        allowEditing: true,
        mode: "Auto"
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    this.toolbar = ['Edit'];
}
}

```

## MAIN.TS

```

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

```

**Note:** When the edit mode is set to **Auto**, you can change the cells to editable mode by double-clicking anywhere at the TreeGrid and edit the task details in the edit dialog by double-clicking anywhere at the chart.

### Dialog editing

Modify the task details through dialog by setting edit [mode](#) property as **Dialog**.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [columns]="columns" [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public toolbar?: string[];
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ],
  }
}

```

```

];
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
};
this.editSettings = {
  allowEditing: true,
  mode: "Dialog"
};
this.columns = [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
];
this.toolbar = ['Edit'];
}
}

```

### MAIN.TS

```

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

```

**Note:** In dialog editing mode, the edit dialog will appear while performing double-click action in both TreeGrid and chart sides.

#### Taskbar editing

Modify the task details through user interaction such as resizing and dragging the taskbar by enabling the [allowTaskbarEditing](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,

```

```

    selector: 'app-root',
    template:
      `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [editSettings]="editSettings"
[columns]="columns" [toolbar]="toolbar"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public columns?: object[];
    public toolbar?: string[];
    public editSettings?: EditSettingsModel;
    public ngOnInit(): void {
      this.data = [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ];
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      };
      this.editSettings = {
        allowTaskbarEditing: true
      };
    }
  }

```



```

        this.columns = [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' },
        ];
        this.toolbar = ['Edit'];
    }
}

```

## MAIN.TS

```

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

```

### Dependency editing

Modify the task dependencies using mouse interactions by enabling the [allowTaskbarEditing](#) property along with mapping the task dependency data source field to the [dependency](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [editSettings]="editSettings"
[columns]="columns" [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public toolbar?: string[];
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {

```

```

        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
this.editSettings = {
    allowTaskbarEditing:true
};
this.columns = [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250' },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
];
this.toolbar = ['Edit'];
}
}

```

**MAIN.TS**

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

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

### Enabling predecessors or task relationships

Predecessor or task dependency in the Gantt component is used to depict the relationship between the tasks.

Start to Start (SS) : You cannot start a task until the dependent task starts.

Start to Finish (SF) : You cannot finish a task until the dependent task finishes.

Finish to Start (FS) : You cannot start a task until the dependent task completes.

Finish to Finish (FF) : You cannot finish a task until the dependent task completes.

You can show the relationship in tasks, by using the [dependency](#) property as shown in the following code example:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DayMarkersService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [DayMarkersService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
```

```

    ]
  },
  {
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 6, TaskName: 'Develop floor plan for estimation',
        StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
      { TaskID: 7, TaskName: 'List materials', StartDate: new
        Date('04/04/2019'), Duration: 3, Progress: 50 },
      { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
        Date('04/04/2019'), Duration: 0, Predecessor: "6SS", Progress: 50 }
    ]
  },
];
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
};
}
}

```

## MAIN.TS

```

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

```

## Assigning resources

You can display and assign the resource for each task in the Gantt control. Create a collection of JSON object, which contains id, name, unit and group of the resources and assign it to the [resources](#) property.

Map these fields to the Gantt control using the [resourceFields](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',

```

```

    template:
      `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [treeColumnIndex]="1"
[highlightWeekends]="true" [labelSettings]="labelSettings"
[resourceFields]="resourceFields" [resources]="resources"></ejs-gantt>`,
      encapsulation: ViewEncapsulation.None
    })
  export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public resources?: object[];
    public taskSettings?: object;
    public labelSettings?: object;
    public resourceFields?: object;
    public ngOnInit(): void {
      this.data = [
        {
          TaskID: 1,
          TaskName: 'Project initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
            {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
            {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          subtasks: [
            {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
            {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
            {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]
            }
          ]
        }
      ]
    };
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      resourceInfo: 'resources',

```

```

        child: 'subtasks'
    };
    this.labelSettings = {
        leftLabel: 'TaskName',
        rightLabel: 'resources'
    };
    this.resources = [
        { resourceId: 1, resourceName: 'Martin Tamer' },
        { resourceId: 2, resourceName: 'Rose Fuller' },
        { resourceId: 3, resourceName: 'Margaret Buchanan' },
        { resourceId: 4, resourceName: 'Fuller King' },
        { resourceId: 5, resourceName: 'Davolio Fuller' },
        { resourceId: 6, resourceName: 'Van Jack' },
        { resourceId: 7, resourceName: 'Fuller Buchanan' },
        { resourceId: 8, resourceName: 'Jack Davolio' },
        { resourceId: 9, resourceName: 'Tamer Vinet' },
        { resourceId: 10, resourceName: 'Vinet Fuller' },
        { resourceId: 11, resourceName: 'Bergs Anton' },
        { resourceId: 12, resourceName: 'Construction Supervisor' }
    ];
    this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName'
    };
}

```

## MAIN.TS

```

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

```

## Enable filtering

The filtering feature enables you to view reduced amount of records based on filter criteria. Gantt provides support for menu filtering support for each columns. It can be enabled by setting the [allowFiltering](#) property to true along with injecting the `Filter` module module as shown in the following code example. Filtering feature can also be customized using the [filterSettings](#) property.

To use filter, inject the `FilterService` in the provider section of `AppModule`.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService, ToolbarService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem } from '@syncfusion/ej2-angular-gantt';
@Component({
    imports: [
        GanttModule
    ],

```

```

providers: [FilterService, ToolbarService],
standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [allowFiltering]='true'
[dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
[toolbar]="toolbar" [timelineSettings]="timelineSettings"
[labelSettings]="labelSettings" [projectStartDate]="projectStartDate"
[projectEndDate]="projectEndDate"
[splitterSettings]="splitterSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public timelineSettings?: object;
  public labelSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  public toolbar?: ToolbarItem[];
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',

```

```

        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.columns = [
        { field: 'TaskName', headerText: 'Task Name', width: '250' ,
clipMode: 'EllipsisWithTooltip' },
        { field: 'StartDate', headerText: 'Start Date' },
        { field: 'Duration', headerText: 'Duration' },
        { field: 'EndDate', headerText: 'End Date' },
        { field: 'Predecessor', headerText: 'Predecessor' }
    ];
    this.timelineSettings = {
        timelineUnitSize: 70,
        topTier: {
            format: 'MMM dd, yyyy',
            unit: 'Day',
        },
        bottomTier: {
            unit: 'Hour',
            format: 'hh : mm a'
        }
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.toolbar = ['Search'];
    this.labelSettings = {
        leftLabel: 'TaskName',
    };
    this.projectStartDate = new Date('07/16/1969 01:00:00 AM');
    this.projectEndDate = new Date('07/25/1969');
    }
}

```

## MAIN.TS

```

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

```

## Enable sorting

The sorting feature enables you to order the records. It can be enabled by setting the [allowSorting](#) property to **true**. Provide the **Sort** module as follows. If **Sort** module is not provided, you cannot sort when a header is clicked. The sorting feature can be customized using the `sortSettings(../api/gantt/sortSettings/)` property.

To use sort, inject the [SortService](#) in the provider section of **AppModule**.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```



```

import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [splitterSettings]="splitterSettings" [allowSorting]= 'true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    }
}

```

### MAIN.TS

```

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

```

### Defining eventmarkers

The [eventMarkers](#) property in Gantt component is used to highlight the important event in Gantt chart part. By using this feature, you can add the lines and label to highlight important days in your project.

To highlight the days, inject the `DayMarkersService` in the provider section of `AppModule`.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DayMarkersService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { projectNewData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    providers: [DayMarkersService],
    standalone: true,
    selector: 'app-root',
    template:
`<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [eventMarkers]="eventMarkers"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    // Data for Gantt

```

```

public data?: object[];
public taskSettings?: object;
public eventMarkers?: object[];
public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.eventMarkers = [
        {
            day: new Date('04/09/2019'),
            label: 'Research phase'
        }, {
            day: new Date('04/30/2019'),
            label: 'Design phase'
        }, {
            day: new Date('05/23/2019'),
            label: 'Production phase'
        }, {
            day: new Date('06/20/2019'),
            label: 'Sales and marketing phase'
        }
    ];
}
}

```

### **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

Use the following command to run the application in the web browser:

```

`javascript
ng serve --open
`

```

The following example shows a basic Gantt:

### **APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';

```

```

import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" ></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
  }
};

```

```
}
}
```

### MAIN.TS

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

### Module in Angular Gantt component

The modules that are available in Gantt are as follows.

Module	Description
----- -----	
<a href="#">Sort</a>	Inject this module to use sorting feature.
<a href="#">Filter</a>	Inject this module to use filtering feature.
<a href="#">Reorder</a>	Inject this module to use reorder feature.
<a href="#">ExcelExport</a>	Inject this module to use excel export feature.
<a href="#">PdfExport</a>	Inject this module to use PDF export feature.
<a href="#">RowDD</a>	Inject this module to use row drag and drop feature.
<a href="#">Resize</a>	Inject this module to use resize feature.
<a href="#">Toolbar</a>	Inject this module to use toolbar feature.
<a href="#">Edit</a>	Inject this module is use editing feature.
<a href="#">Selection</a>	Inject this module to use selection feature.
<a href="#">DayMarkers</a>	Inject this module to use event markers.
<a href="#">ContextMenu</a>	Inject this module to use context menu feature.
<a href="#">ColumnMenu</a>	Inject this module to use column menu feature.
<a href="#">VirtualScroll</a>	Inject this module to use virtual scroll feature.
<a href="#">CriticalPath</a>	Inject this module to use critical path feature.

These modules should be injected into the Gantt using the **Gantt.Inject** method.

### Data binding in Angular Gantt component

The Gantt uses **DataManager**, which supports both RESTful JSON data services binding and local JavaScript object array binding. The [Link to the Video](#) property can be assigned either with the instance of DataManager or JavaScript object array collection. Gantt provides support to bind two kinds of data,

- Local data
- Remote data

The following video explains the data binding in Gantt chart :

### Local data

To bind local data to Gantt, you can assign a JavaScript object array to the [dataSource](#) property. The local data source can also be provided as an instance of the [DataManager](#).

In local data binding, the data source for rendering the Gantt component is retrieved from the same application locally.

The following are the two types of data binding possible with the Gantt component:

- Hierarchical data binding.
- Self-referential data binding (Flat data).

### Hierarchical data binding

The [child](#) property is used to map the child records in hierarchical data.

The following code example shows how to bind the hierarchical local data into the Gantt component.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
    ],
  }
```

```

        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
};
}
}

```

### MAIN.TS

```

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

```

### Self-referential data binding (Flat data)

The Gantt component can be bound with self-referential data by mapping the data source field values to the [id](#) and [parentID](#) properties.

- ID field: This field contains unique values used to identify each individual task and it is mapped to the [id](#) property.
- Parent ID field: This field contains values that indicate parent tasks and it is mapped to the [parentID](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template:
      `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public ngOnInit(): void {
      this.data = [
        { TaskID: 1, TaskName: 'Project Initiation', StartDate: new
Date('04/02/2019'), EndDate: new Date('04/21/2019') },
        { TaskID: 2, TaskName: 'Identify Site location', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, ParentId: 1 },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, ParentId: 1 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, ParentId: 1 },
        { TaskID: 5, TaskName: 'Project Estimation', StartDate: new
Date('04/02/2019'), EndDate: new Date('04/21/2019') },
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, ParentId: 2 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, ParentId: 2 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, ParentId: 2 }
      ];
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        parentID: 'ParentId'
      };
    }
  }
}

```

## 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

To bind remote data to the Gantt component, assign service data as an instance of **DataManager** to the [dataSource](#) property.

## APP.COMPONENT.TS

```

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

```



```

import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: DataManager;
  public taskSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://ej2services.syncfusion.com/production/web-
services/api/GanttData',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });
    this.taskSettings = {
      id: 'TaskId',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'SubTasks'
    };
    this.columns = [
      { field: 'TaskName', headerText: 'Task Name', width: '250',
clipMode: 'EllipsisWithTooltip' },
      { field: 'StartDate' },
      { field: 'Duration' }
    ];
  }
}

```

### MAIN.TS

```

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

```

### URL Adaptor

In Gantt, we can fetch data from SQL database using ADO.NET Entity Data Model and update the changes on CRUD action to the server by using DataManager support. To communicate with the

remote data we are using `UrlAdaptor` of `DataManager` property to call the server method and get back resultant data in JSON format. We can know more about `UrlAdaptor` from [here](#).

Please refer the [link](#) to create the `ADO.NET` Entity Data Model in Visual studio,

We can define data source for Gantt as instance of `DataManager` using `url` property of `DataManager`.

Please Check the below code snippet to assign data source to Gantt.

```
`typescript
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';

@Component({
  selector: 'app-root',
  template:
    <ejs-gantt id="ganttDefault" height="430px" [dataSource]="data" [taskFields]="taskSettings"
    [columns]="columns"></ejs-gantt>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data: DataManager;
  public taskSettings: object;
  public columns: object[];
  public ngOnInit(): void {
    this.data = new DataManager({
      url: '/Home/UrlDatasource',
      adaptor: new UrlAdaptor
    });
    this.taskSettings = {
      id: 'TaskId',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'SubTasks'
    }
  }
}
```

```

};
this.columns = [
{ field: 'TaskName', headerText: 'Task Name', width: '250', clipMode: 'EllipsisWithTooltip' },
{ field: 'StartDate' },
{ field: 'Duration' }
];
}
}
,

```

```

`typescript
GanttDataSourceEntities db = new GanttDataSourceEntities();
public ActionResult UrlDatasource(DataManagerRequest dm)
{
List<GanttData>DataList = db.GanttDatas.ToList();
var count = DataList.Count();
return Json(new { result = DataList, count = count });
}
,

```

#### *Load child on demand*

To render child records on demand, assign a remote service URL in the instance of `DataManager` to the `Url` property. To interact with the remote data source, provide the endpoint URL and also define the [hasChildMapping](#) property in taskFields of Gantt Chart.

The `hasChildMapping` property maps the field name in the data source, which denotes whether the current record holds any child records. This is useful internally to show expand icon while binding child data on demand.

When [loadChildOnDemand](#) is disabled, all the root nodes are rendered in a collapsed state at initial load. On expanding the root node, the child nodes will be loaded from the remote server.

When `enableVirtualization` is enabled and `loadChildOnDemand` is disabled, only the current viewport root nodes are rendered in a collapsed state.

When a root node is expanded, its child nodes are rendered and maintained in a collection locally, such that on consecutive expand/collapse actions on the root node, the child nodes are loaded locally instead of from the remote server.

When the `loadChildOnDemand` is enabled, parent records are rendered in an expanded state.

```

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

```

```

import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { VirtualScrollService, SelectionService } from '@syncfusion/ej2-angular-gantt';

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

  template: `<ejs-gantt id="ganttdefault" [dataSource]="data" [taskFields]="taskSettings"
[loadChildOnDemand]="false" [enableVirtualization]="true" [allowSelection]="true"
[labelSettings]="labelSettings" [columns]="columns" [treeColumnIndex]="1" height="450px"
[projectStartDate]="projectStartDate" [projectEndDate]="projectEndDate"
[highlightWeekends]="true" [taskbarHeight]="20" [rowHeight]="40" [tooltipSettings]="tooltipSettings"
[splitterSettings]="splitterSettings">
</ejs-gantt>`,

  providers: [VirtualScrollService, SelectionService]
})

export class AppComponent implements OnInit {
  public data: object;
  public taskSettings: object;
  public columns: object[];
  public labelSettings: object;
  public splitterSettings: object;
  public tooltipSettings: object;
  public projectStartDate: Date;
  public projectEndDate: Date;

  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://services.syncfusion.com/angular/production/api/GanttLoadOnDemand',
      adaptor: new WebApiAdaptor,
      crossDomain: true
    });

    this.taskSettings = {
      id: 'taskId',
      name: 'taskName',
      startDate: 'startDate',
      endDate: 'endDate',
      duration: 'duration',

```

```

progress: 'progress',
hasChildMapping: "isParent",
parentID: "parentID"
};
this.tooltipSettings= {
showTooltip: true
};
this.splitterSettings = {
columnIndex: 3
};
this.columns = [
{ field: 'taskId', width:80 },
{ field: 'taskName', headerText: 'Name', width: '200', clipMode: 'EllipsisWithTooltip' },
{ field: 'startDate' },
{ field: 'duration' },
{ field: 'progress' },
];
this.projectStartDate = new Date('01/02/2000');
this.projectEndDate = new Date('01/06/2002');
}
}
`

```

The following code example describes handling of Load on demand at server end.

```

`typescript
public object Get()
{
DataOperations operation = new DataOperations();
var queryString = Request.Query;
if (tree.Count == 0)
tree = TreeData.GetTree();
if (queryString.Keys.Contains("$filter") && !queryString.Keys.Contains("$stop"))
{
StringValues filter;

```

```
queryString.TryGetValue("$filter", out filter);
int? fltr;
if (filter[0].ToString().Split("eq")[1] == " null")
{
    fltr = null;
}
else
{
    fltr = Int32.Parse(filter[0].ToString().Split("eq")[1]);
}
IQueryable<TreeData> data1 = tree.Where(f => f.parentID == fltr).AsQueryable();
return new { result = data1.ToList(), count = data1.Count() };
}

StringValues expand;
queryString.TryGetValue("$expand", out expand);
if (queryString.Keys.Contains("$expand")) // setting the ExpandStateMapping property whether is true
or false
{
    if (expand[0].ToString().Split(",")[0] == "ExpandingAction")
    {
        var val = TreeData.GetTree().Where(ds => ds.taskId ==
int.Parse(expand[0].ToString().Split(",")[1])).FirstOrDefault();
        val.IsExpanded = true;
    }
    else if (expand[0].ToString().Split(",")[0] == "CollapsingAction")
    {
        var val = TreeData.GetTree().Where(ds => ds.taskId ==
int.Parse(expand[0].ToString().Split(",")[1])).FirstOrDefault();
        val.IsExpanded = false;
    }
}

List<TreeData> data = tree.ToList();
if (queryString.Keys.Contains("$select"))
{
```

```
data = (from ord in tree
select new TreeData
{
    parentID = ord.parentID
}
).ToList();
return data;
}

data = data.Where(p => p.parentID == null).ToList();
int count = data.Count;
if (queryString.Keys.Contains($"$inlinecount"))
{
    StringValues Skip;
    StringValues Take;
    StringValues loadchild;
    int skip = (queryString.TryGetValue("$skip", out Skip)) ? Convert.ToInt32(Skip[0]) : 0;
    int top = (queryString.TryGetValue("$top", out Take)) ? Convert.ToInt32(Take[0]) : data.Count();
    var GroupData = TreeData.GetTree().ToList().GroupBy(rec => rec.parentID)
        .Where(g => g.Key != null).ToDictionary(g => g.Key?.ToString(), g => g.ToList());
    foreach (var Record in data.ToList())
    {
        if (GroupData.ContainsKey(Record.taskId.ToString()))
        {
            var ChildGroup = GroupData[Record.taskId.ToString()];
            if (ChildGroup?.Count > 0)
                AppendChildren(ChildGroup, Record, GroupData, data);
        }
    }
    if (expand.Count > 0 && expand[0].ToString().Split(",")[0] == "CollapsingAction")
    {
        string IdMapping = "taskId";
        List<WhereFilter> CollapseFilter = new List<WhereFilter>();
    }
}
```

```

CollapseFilter.Add(new WhereFilter() { Field = IdMapping, value = expand[0].ToString().Split(",")[1],
Operator = "equal" });
var CollapsedParentRecord = operation.PerformFiltering(data, CollapseFilter, "and");
var index = data.Cast<object>().ToList().IndexOf(CollapsedParentRecord.Cast<object>().ToList()[0]);
skip = index;
}
else if (expand.Count > 0 && expand[0].ToString().Split(",")[0] == "ExpandingAction")
{
string IdMapping = "taskId";
List<WhereFilter> ExpandFilter = new List<WhereFilter>();
ExpandFilter.Add(new WhereFilter() { Field = IdMapping, value = expand[0].ToString().Split(",")[1],
Operator = "equal" });
var ExpandedParentRecord = operation.PerformFiltering(data, ExpandFilter, "and");
var index = data.Cast<object>().ToList().IndexOf(ExpandedParentRecord.Cast<object>().ToList()[0]);
skip = index;
}
return new { result = data.Skip(skip).Take(top), count = data.Count };
}
else
{
return TreeData.GetTree();
}
}

private void AppendChildren(List<TreeData> ChildRecords, TreeData ParentItem, Dictionary<string,
List<TreeData>> GroupData, List<TreeData> data)
{
var queryString = Request.Query;
string TaskId = ParentItem.taskId.ToString();
var index = data.IndexOf(ParentItem);
foreach (var Child in ChildRecords)
{
string ParentId = Child.parentID.ToString();
if (TaskId == ParentId && (bool)ParentItem.IsExpanded)
{

```



```
if (data.IndexOf(Child) == -1)
((IList)data).Insert(++index, Child);
if (GroupData.ContainsKey(Child.taskId.ToString()))
{
var DeepChildRecords = GroupData[Child.taskId.ToString()];
if (DeepChildRecords?.Count > 0)
AppendChildren(DeepChildRecords, Child, GroupData, data);
}
}
}
}
// GET: api/Orders/
[HttpGet("{id}", Name = "Get")]
public string Get(int id)
{
return "value";
}
[HttpPost]
public object Post([FromBody] TreeData[] value)
{
//handle insert action
for (var i = 0; i < value.Length; i++)
{
tree.Insert(0, value[i]);
}
return value;
}
//// PUT: api/Orders
[HttpPut]
public object Put([FromBody] TreeData[] value)
{
//handle edit action
if (value.Length == 1 && value[0].isParent == true)
```

```
{
UpdateDependentRecords(value[0]);
}
for (var i = 0; i < value.Length; i++) {
var ord = value[i];
TreeData val = tree.Where(or => or.taskId == ord.taskId).FirstOrDefault();
val.taskId = ord.taskId;
val.taskName = ord.taskName;
val.endDate = ord.endDate;
val.startDate = ord.startDate;
val.duration = ord.duration;
val.predecessor = ord.predecessor;
}
return value;
}

private void UpdateDependentRecords(TreeData ParentItem)
{
var data = tree.Where(p => p.parentID == ParentItem.taskId).ToList();
var previousData = tree.Where(p => p.taskId == ParentItem.taskId).ToList();
var previousStartDate = previousData[0].startDate;
var previousEndDate = previousData[0].endDate;
double sdiff = (double)GetTimeDifference((DateTime)previousStartDate,
(DateTime)ParentItem.startDate);
double ediff = (double)GetTimeDifference((DateTime)previousEndDate,
(DateTime)ParentItem.endDate);
GetRootChildRecords(ParentItem);
for(var i=0; i<ChildRecords.Count;i++)
{
ChildRecords[i].startDate = ((DateTime)ChildRecords[i].startDate).AddSeconds(sdiff);
ChildRecords[i].endDate = ((DateTime)ChildRecords[i].endDate).AddSeconds(ediff);
}
}

private void GetRootChildRecords(TreeData ParentItem)
{

```

```

var currentchildRecords = tree.Where(p => p.parentID == ParentItem.taskId).ToList();
for (var i = 0; i < currentchildRecords.Count; i++) {
    var currentRecord = currentchildRecords[i];
    ChildRecords.Add(currentRecord);
    if (currentRecord.isParent == true)
    {
        GetRootChildRecords(currentRecord);
    }
}

public object GetTimeDifference(DateTime sdate, DateTime edate)
{
    return new DateTime(edate.Year, edate.Month, edate.Day, edate.Hour, edate.Minute, edate.Second,
        DateTimeKind.Utc).Subtract(new DateTime(sdate.Year, sdate.Month, sdate.Day, sdate.Hour,
        sdate.Minute, sdate.Second, DateTimeKind.Utc)).TotalSeconds;
}

// DELETE: api/ApiWithActions
[HttpDelete("{id:int}")]
[Route("Orders/{id:int}")]
public object Delete(int id)
{
    //handle delete action
    tree.Remove(tree.Where(or => or.taskId == id).FirstOrDefault());
    return Json(id);
}

public class CRUDModel<T> where T : class
{
    public TreeData Value;
    public int Key { get; set; }
    public int RelationalKey { get; set; }
    public List<TreeData> added { get; set; }
    public List<TreeData> changed { get; set; }
    public List<TreeData> deleted { get; set; }
}

```

```
public class TreeData
{
    public static List<TreeData> tree = new List<TreeData>();
    [System.ComponentModel.DataAnnotations.Key]
    public int taskId { get; set; }
    public string taskName { get; set; }
    public DateTime startDate { get; set; }
    public DateTime endDate { get; set; }
    public string duration { get; set; }
    public int progress { get; set; }
    public int? parentID { get; set; }
    public string predecessor { get; set; }
    public bool? isParent { get; set; }
    public bool? IsExpanded { get; set; }
    public static List<TreeData> GetTree()
    {
        if (tree.Count == 0)
        {
            Random rand = new Random();
            var x = 0;
            int duration = 0;
            DateTime startDate = new DateTime(2000, 1, 3, 08, 00, 00);
            for (var i = 1; i <= 50; i++)
            {
                startDate = startDate.AddDays(i == 1 ? 0 : 7);
                DateTime childStartDate = startDate;
                TreeData Parent = new TreeData()
                {
                    taskId = ++x,
                    taskName = "Task " + x,
                    startDate = startDate,
                    endDate = childStartDate.AddDays(26),
                    duration = "20",
```

```

progress = rand.Next(100),
predecessor = null,
isParent = true,
parentID = null,
IsExpanded = false
};
tree.Add(Parent);
for (var j = 1; j <= 4; j++)
{
childStartDate = childStartDate.AddDays(j == 1 ? 0 : duration + 2);
duration = 5;
tree.Add(new TreeData()
{
taskId = ++x,
taskName = "Task " + x,
startDate = childStartDate,
endDate = childStartDate.AddDays(5),
duration = duration.ToString(),
progress = rand.Next(100),
parentID = Parent.taskId,
predecessor = (j > 1 ? (x - 1) + "FS" : ""),
isParent = false,
IsExpanded = false
});
}
}
}
return tree;
}
}
`

```

### *Limitations*

- Filtering, sorting and searching are not supported in load on demand.

- Only Self-Referential type data is supported with remote data binding in Gantt Chart.
- Load-on-demand supports only the validated data source

#### *Sending additional parameters to the server*

We can pass additional parameters using [addParams](#) method of [Query](#) class. In server side we have inherited and shown the additional parameter value in Syncfusion DataManager class itself. We pass an additional parameter in load time using [load](#) event. We can also pass additional parameter to the CRUD model. Please Check the below code snippet to send additional parameter to Gantt.

```
`typescript
```

```
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt, ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-gantt';
import { DataManager, UrlAdaptor, Query } from '@syncfusion/ej2-data';

@Component({
  selector: 'app-root',
  template:
    <ejs-gantt id="ganttDefault" height="430px" [dataSource]="data" [taskFields]="taskSettings"
    [editSettings]="editSettings" [toolbar]="toolbar" (load)="load($event)"></ejs-gantt>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data: DataManager;
  public taskSettings: object;
  public editSettings: EditSettingsModel;
  public toolbar: ToolbarItem[];
  public columns: object[];
  @ViewChild('gantt', {static: true})
  public ganttObj: GanttComponent;
  public ngOnInit(): void {
    this.data = new DataManager({
      url: 'http://localhost:50039/Home/UrlDatasource',
      adaptor: new UrlAdaptor,
      batchUrl: 'http://localhost:50039/Home/BatchSave',
    });
    this.taskSettings = {
```

```

id: 'TaskId',
name: 'TaskName',
startDate: 'StartDate',
duration: 'Duration',
progress: 'Progress',
dependency: 'Predecessor',
child: 'SubTasks'
};

this.columns = [
{ field: 'TaskName', headerText: 'Task Name', width: '250', clipMode: 'EllipsisWithTooltip' },
{ field: 'StartDate' },
{ field: 'Duration' }
];

this.editSettings = {
allowAdding: true,
allowEditing: true,
allowDeleting: true
};

this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'ExpandAll', 'CollapseAll'];
load: function(args) {
this.ganttObj.query = new Query().addParams('ej2Gantt', "test");
}
}
,
`typescript
namespace URLAdaptor.Controllers
{
public class HomeController : Controller
{
...///
//inherit the class to show age as property of DataManager
public class Test : DataManagerRequest

```

```

{
public string ej2Gantt { get; set; }
}
public ActionResult UrlDatasource([FromBody]Test dm)
{
if (DataList == null)
{
ProjectData datasource = new ProjectData();
DataList = datasource.GetUrlDataSource();
}
var count = DataList.Count();
return Json(new { result = DataList, count = count }, JsonRequestBehavior.AllowGet);
}
...///
public class ICRUDModel<T> where T : class
{
public object key { get; set; }
public T value { get; set; }
public List<T> added { get; set; }
public List<T> changed { get; set; }
public List<T> deleted { get; set; }
public IDictionary<string, object> @params { get; set; }
}
...///
}
}
,

```

You can find the full sample from [here](#).

#### Handling HTTP error

During server interaction from the Gantt, some server-side exceptions may occur, and you can acquire those error messages or exception details in client-side using the [actionFailure](#) event.

The argument passed to the `actionFailure` event contains the error details returned from the server.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
```



```

import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { DataManager } from '@syncfusion/ej2-data';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings"
    (actionFailure)="actionFailure($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  actionFailure = ($event: any) => {
    let span: HTMLElement = document.createElement('span');
    this.ganttObj!.element.parentNode!.insertBefore(span,
    this.ganttObj!.element);
    span.style.color = '#FF0000'
    span.innerHTML = 'Server exception: 404 Not found';
  }
  // Data for Gantt
  public data?: DataManager;
  public taskSettings?: object;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent;
  public ngOnInit(): void {
    this.data = new DataManager({
      url: 'http://some.com/invalidUrl',
    });
    this.taskSettings = {
      id: 'TaskId',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'SubTasks'
    };
  }
}

```

## 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 Fetch

You can use Gantt [dataSource](#) property to bind the data source to Gantt from external Fetch request. In the below code we have fetched the data source from the server with the help of Fetch request and provided that to [dataSource](#) property by using [onSuccess](#) event of the Fetch.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { Fetch } from '@syncfusion/ej2-base/src/fetch';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { DataManager, WebApiAdaptor, UrlAdaptor } from '@syncfusion/ej2-
data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button ej2-button id='binddata' (click)='bind()'>Bind Data</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" [dataSource]="data"
    height="430px" [taskFields]="taskSettings"
    [projectStartDate]="projectStartDate"
    [projectEndDate]="projectEndDate"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  @ViewChild('gantt', { static: true })
  public ganttObj?: GanttComponent;
  public gantt?: GanttComponent;
  public temp: any;
  public ngOnInit(): void {
    this.temp = this,
    this.data = [],
    this.taskSettings = {
      id: 'TaskId',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'SubTasks'
    },
    this.projectStartDate = new Date('02/24/2019'),
    this.projectEndDate = new Date('07/20/2019')
```

```

    }
    bind(): void {
        const temp = this.ganttObj;
        let fetch = new
Fetch("https://ej2services.syncfusion.com/production/web-
services/api/GanttData", "GET");
        temp!.showSpinner();
        fetch.send();
        fetch.onSuccess = function (data: any) {
            temp!.hideSpinner();
            temp!.dataSource = data.Items;
            temp!.refresh();
        };
    };
}

```

### MAIN.TS

```

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

```

Note: If you bind the dataSource from this way, then it acts like a local dataSource. So you cannot perform any server side crud actions.

### Split task

The **Split-task** feature allows you to split a task or interrupt the work during planned or unforeseen circumstances.

We can split the task either in load time or dynamically, by defining the segments either in hierarchical or self-referential way.

### Hierarchical

To split a task at load time in hierarchical way, we need to define the segment details in datasource and this field should be mapped by using the [taskFields.segments](#) property.

`typescript

```

[
{
TaskID: 1, TaskName: 'Identify Site location', StartDate: new Date('04/02/2019'), Duration: 4, Progress:
50,
Segments: [
{ StartDate: new Date("04/02/2019"), Duration: 2 },
{ StartDate: new Date("04/04/2019"), Duration: 2 }
]
}
]

```

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { EditSettingsModel, ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
    [taskFields]="taskSettings">
      </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50,
          Segments: [
            { StartDate: new Date("04/02/2019"), Duration: 2 },
            { StartDate: new Date("04/04/2019"), Duration: 2 }
          ] },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4 , Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },

```

```

        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
}
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks',
    segments: 'Segments'
};
}
}

```

### MAIN.TS

```

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

```

### Self-referential

We can also define segment details as a flat data and this collection can be mapped by using [segmentData](#) property. The segment id field of this collection is mapped by using the [taskFields.segmentId](#) property.

`typescript

```

taskFields: {
    segmentId: "segmentId"
},
segmentData: [
    { segmentId: 1, StartDate: new Date("02/04/2019"), Duration: 2 },
    { segmentId: 1, StartDate: new Date("02/05/2019"), Duration: 5 },
    { segmentId: 4, StartDate: new Date("04/02/2019"), Duration: 2 },
    { segmentId: 4, StartDate: new Date("04/04/2019"), Duration: 2 }
],
`

```

### APP.COMPONENT.TS

```

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

```

```

import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { EditSettingsModel, ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
    [taskFields]="taskSettings" [segmentData]="segmentData"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  segmentData: { segmentId: number; StartDate: Date; Duration: number; }[]
  | undefined;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
        segmentId: 'segmentId'
    };
    this.segmentData = [
        { segmentId: 2, StartDate: new Date("04/02/2019"), Duration: 2 },
        { segmentId: 2, StartDate: new Date("04/04/2019"), Duration: 2 },
        { segmentId: 4, StartDate: new Date("04/02/2019"), Duration: 2 },
        { segmentId: 4, StartDate: new Date("04/04/2019"), Duration: 2 }
    ];
    }
}

```

### MAIN.TS

```

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

```

Note: Segment id field contains id of a task which should be split at load time.

### Improve performance by disabling validations

The [autoCalculateDateScheduling](#) property can help you reduce the time taken for the Gantt chart to render on the initial load. When this API is enabled, parent-child validation, data validation, and predecessor validation are restricted, allowing the Gantt chart to load more quickly. Since we are disabling the validations, data source provided to gantt should have all data such as start date, end date, duration, as proper data.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { VirtualScrollService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { GanttComponent, VirtualScrollService } from '@syncfusion/ej2-angular-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [VirtualScrollService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
    [taskFields]="taskSettings" [treeColumnIndex]="1"

```

```

[splitterSettings]="splitterSettings" [columns]="columns"
[labelSettings]="labelSettings"
[allowSelection]="true" [enableVirtualization]="true"
[autoCalculateDateScheduling]="false" [editSettings] = "editSettings"
[highlightWeekends]="true"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public columns?: object[];
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItem[];
    public labelSettings?: object;
    public ngOnInit(): void {
        let tempData: any[] = [
            {
                TaskID: 1, TaskName: 'Product concept',StartDate: new
                Date('04/02/2019'), EndDate: new Date('04/21/2019'),
                parentID: 0
            },
            {
                TaskID: 2, TaskName: 'Defining the product and its usage',
                StartDate: new Date('04/02/2019'),
                Duration: 3, Progress: 30, parentID: 1
            },
            {
                TaskID: 3, TaskName: 'Defining target audience', StartDate: new
                Date('04/02/2019'),
                parentID: 1, Duration: 3
            },
            {
                TaskID: 4, TaskName: 'Prepare product sketch and notes', StartDate:
                new Date('04/05/2019'),
                Duration: 2, parentID: 1, Progress: 30
            },
            {
                TaskID: 5, TaskName: 'Concept approval', StartDate: new
                Date('04/08/2019'),
                parentID: 0, Duration: 0
            },
            {
                TaskID: 6, TaskName: 'Market research', StartDate: new
                Date('04/02/2019'),
                parentID: 0, EndDate: new Date('04/21/2019')
            },
            {
                TaskID: 7, TaskName: 'Demand analysis', StartDate: new
                Date('04/04/2019'),
                EndDate: new Date('04/21/2019'), parentID: 6
            },
            {
                TaskID: 8, TaskName: 'Customer strength', StartDate: new
                Date('04/09/2019'),
                Duration: 4, parentID: 7, Progress: 30
            }
        ]
    }
}

```



```

    },
    {
        TaskID: 9, TaskName: 'Market opportunity analysis', StartDate: new
Date('04/09/2019'),
        Duration: 4, parentID: 7
    },
    {
        TaskID: 10, TaskName: 'Competitor analysis', StartDate: new
Date('04/15/2019'),
        Duration: 4, parentID: 6, Progress: 30
    },
    {
        TaskID: 11, TaskName: 'Product strength analysis', StartDate: new
Date('04/15/2019'),
        Duration: 4, parentID: 6
    },
    {
        TaskID: 12, TaskName: 'Research complete', StartDate: new
Date('04/18/2019'),
        Duration: 0, parentID: 6
    },
    {
        TaskID: 13, TaskName: 'Product design and development', StartDate:
new Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019')
    },
    {
        TaskID: 14, TaskName: 'Functionality design', StartDate: new
Date('04/19/2019'),
        Duration: 3, parentID: 13, Progress: 30
    },
    {
        TaskID: 15, TaskName: 'Quality design', StartDate: new
Date('04/19/2019'),
        Duration: 3, parentID: 13
    },
    {
        TaskID: 16, TaskName: 'Define reliability', StartDate: new
Date('04/24/2019'),
        Duration: 2, Progress: 30, parentID: 13
    },
    {
        TaskID: 17, TaskName: 'Identifying raw materials', StartDate: new
Date('04/24/2019'),
        Duration: 2, parentID: 13
    },
    {
        TaskID: 18, TaskName: 'Define cost plan', StartDate: new
Date('04/04/2019'),
        parentID: 13, EndDate: new Date('04/21/2019')
    },
    {
        TaskID: 19, TaskName: 'Manufacturing cost', StartDate: new
Date('04/26/2019'),
        Duration: 2, Progress: 30, parentID: 18
    },
    {

```

```

    TaskID: 20, TaskName: 'Selling cost', StartDate: new
Date('04/26/2019'),
    Duration: 2, parentID: 18
  },
  {
    TaskID: 21, TaskName: 'Development of the final design', StartDate:
new Date('04/30/2019'),
    parentID: 13, EndDate: new Date('04/21/2019')
  },
  {
    TaskID: 22, TaskName: 'Defining dimensions and package volume',
StartDate: new Date('04/30/2019'),
    Duration: 2, parentID: 21, Progress: 30
  },
  {
    TaskID: 23, TaskName: 'Develop design to meet industry standards',
StartDate: new Date('05/02/2019'),
    Duration: 2, parentID: 21
  },
  {
    TaskID: 24, TaskName: 'Include all the details', StartDate: new
Date('05/06/2019'),
    Duration: 3, parentID: 21
  },
  {
    TaskID: 25, TaskName: 'CAD computer-aided design', StartDate: new
Date('05/09/2019'),
    Duration: 3, parentID: 13, Progress: 30
  },
  {
    TaskID: 26, TaskName: 'CAM computer-aided manufacturing', StartDate:
new Date('09/14/2019'),
    Duration: 3, parentID: 13
  },
  {
    TaskID: 27, TaskName: 'Design complete', StartDate: new
Date('05/16/2019'),
    Duration: 0, parentID: 13
  },
  {
    TaskID: 28, TaskName: 'Prototype testing', StartDate: new
Date('05/17/2019'),
    Duration: 4, Progress: 30, parentID: 0
  },
  {
    TaskID: 29, TaskName: 'Include feedback', StartDate: new
Date('05/17/2019'),
    Duration: 4, parentID: 0
  },
  {
    TaskID: 30, TaskName: 'Manufacturing', StartDate: new
Date('05/23/2019'),
    Duration: 5, Progress: 30, parentID: 0
  },
  {
    TaskID: 31, TaskName: 'Assembling materials to finsihed goods',
StartDate: new Date('05/30/2019'),

```

```

        Duration: 5, parentID: 0
    },
    {
        TaskID: 32, TaskName: 'Feedback and testing', StartDate: new
Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
        TaskID: 33, TaskName: 'Internal testing and feedback', StartDate:
new Date('06/06/2019'),
        Duration: 3, parentID: 32, Progress: 45
    },
    {
        TaskID: 34, TaskName: 'Customer testing and feedback', StartDate:
new Date('06/11/2019'),
        Duration: 3, parentID: 32, Progress: 50
    },
    {
        TaskID: 35, TaskName: 'Final product development', StartDate: new
Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
        TaskID: 36, TaskName: 'Important improvements', StartDate: new
Date('06/14/2019'),
        Duration: 4, Progress: 30, parentID: 35
    },
    {
        TaskID: 37, TaskName: 'Address any unforeseen issues', StartDate:
new Date('06/14/2019'),
        Duration: 4, Progress: 30, parentID: 35
    },
    {
        TaskID: 38, TaskName: 'Final product', StartDate: new
Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
        TaskID: 39, TaskName: 'Branding product', StartDate: new
Date('06/20/2019'),
        Duration: 4, parentID: 38
    },
    {
        TaskID: 40, TaskName: 'Marketing and presales', StartDate: new
Date('06/26/2019'), Duration: 4,
        Progress: 30, parentID: 38
    }
];
let virtualData: any[] = [];
let projId: number = 1;
for (let i: number = 0; i < 50; i++) {
    let x: number = virtualData.length + 1;
    let parent: any = {};
    /* tslint:disable:no-string-literal */
    parent['TaskID'] = x;
    parent['TaskName'] = 'Project ' + (i + 1);
    virtualData.push(parent);
}

```

```

    for (let j: number = 0; j < tempData.length; j++) {
      let subtasks: any = {};
      /* tslint:disable:no-string-literal */
      subtasks['TaskID'] = tempData[j].TaskID + x;
      subtasks['TaskName'] = tempData[j].TaskName;
      subtasks['StartDate'] = tempData[j].StartDate;
      subtasks['Duration'] = tempData[j].Duration;
      subtasks['Progress'] = tempData[j].Progress;
      subtasks['parentID'] = tempData[j].parentID + x;
      virtualData.push(subtasks);
    }
  }
  this.data = virtualData,
  this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    parentID: 'parentID'
  };
  this.columns = [
    { field: 'TaskID' },
    { field: 'TaskName' },
    { field: 'StartDate' },
    { field: 'Duration' },
    { field: 'Progress' }
  ];
  this.splitterSettings = {
    columnIndex: 2
  };
  this.editSettings = {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
  },
  this.toolbar = ['Add', 'Cancel', 'CollapseAll', 'Delete', 'Edit',
    'ExpandAll', 'NextTimeSpan', 'PrevTimeSpan', 'Search', 'Update', 'Indent',
    'Outdent'],
  this.labelSettings = {
    leftLabel: 'TaskName',
    taskLabel: 'Progress'
  };
}
}
}

```

### MAIN.TS

```

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

```

### Limitations

Gantt has the support for both Hierarchical and Self-Referential data binding. When rendering the Gantt control with SQL database, we suggest you to use the Self-Referential data binding to maintain the parent-child relation. Because the complex json structure is very difficult to manage it in SQL tables, we need to write a complex queries and we have to write a complex algorithm to find out the proper record details while updating/deleting the inner level task in Gantt data source. We cannot implement both data binding for Gantt control and this is not a recommended way. If both child and parentID are mapped, the records will not render properly because, when task id of a record defined in the hierarchy structure is assigned to parent id of another record, in such case the records will not properly render. As the self-referential will search the record with particular id in flat data only, not in the inner level of records. If we map the parentID field, it will be prioritized and Gantt will be rendered based on the parentID values.

### Observable in Angular Gantt component

An [Observable](#) is used extensively by Angular since it provide significant benefits over techniques for event handling, asynchronous programming, and handling multiple values.

#### Observable binding using Async pipe

Gantt data can be consumed from an [Observable](#) object by piping it through an [async](#) pipe. The [async](#) pipe is used to subscribe the observable object and resolve with the latest value emitted by it.

#### Data binding

The gantt expects an object from the [Observable](#). The emitted value should be an object with properties **result** and **count**.

```
`ts
import { Component, OnInit, ViewChild } from "@angular/core";
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { DataStateChangeEventArgs } from "@syncfusion/ej2-treegrid";
import { Observable } from "rxjs";
import { TaskStoreService } from "../task-store.service";

@Component({
  selector: 'app-root',
  template: `<ejs-gantt #gantt id="ganttContainer"
[dataSource]="tasks | async" [height]="gantHeight" [rowHeight]="25" [allowSorting]="true"
[taskFields]="taskSettings" [editSettings]="editSettings"
[toolbar]="toolbar"

</ejs-gantt>`,
  providers: [TaskStoreService]
})
export class AppComponent implements OnInit {
```

```

public taskSettings: object;
public tasks: Observable<DataStateChangeEventArgs>;
public editSettings: any;
public toolbar: string[];
@ViewChild('gantt')
public ganttChart: GanttComponent;
constructor(private TaskService: TaskStoreService) {
  this.tasks = TaskService;
}
ngOnInit(): void {
  this.taskSettings = {
    id: 'id',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    parentID: 'ParentID',
    resourceInfo: 'resources',
  };
  this.editSettings = {
    allowEditing: true,
    allowAdding: true,
    allowDeleting: true,
    allowTaskbarEditing: true
  };
  this.toolbar = ['Add', 'Edit', 'Delete'];
  const state: any = { skip: 0, take: 10 };
  this.TaskService.execute(state);
}
}
`ts

```

```
import { Injectable } from "@angular/core";
import { Subject, Observable } from "rxjs";
import { HttpClient, HttpHeaders } from "@angular/common/http";
import { TaskModel } from "../task-model";
import { map } from "rxjs/operators";

@Injectable({
  providedIn: "root"
})
export class TaskStoreService extends Subject<any> {
  private apiUrl = "api/tasks";
  constructor(private http: HttpClient) {
    super();
  }
  public execute(state: any): void {
    this.getTasks(state).subscribe(x =>
      super.next(x as any)
    );
  }
  getTasks(state?: any): Observable<TaskModel[]> {
    return this.http.get<TaskModel[]>(this.apiUrl).pipe(
      map(
        (response: any) =>
          <any>{
            result:
              state.take > 0
                ? response.slice(state.skip, state.take)
                : response,
            count: response.length
          }
      )
    );
  }
}
```

Note: Currently, we do not have support to perform CRUD operations using observables in Angular Gantt Chart.

### Columns in Angular Gantt component

In Syncfusion Angular Gantt component, columns are fundamental elements that play a pivotal role in organizing and displaying data within your application. They serve as the building blocks for data presentation, allowing you to specify what data fields to show, how to format and style them, and how to enable various interactions within the gantt.

The column definitions are used as the [dataSource](#) schema in the gantt. The [field](#) property of the [column](#) is necessary to map the data source values in gantt columns.

### Column types

The Syncfusion Gantt component allows you to specify the type of data that a column binds using the `column.type` property. The `type` property is used to determine the appropriate [format](#), such as [number](#) or [date](#), for displaying the column data.

Gantt supports the following column types:

- **string**: Represents a column that binds to string data. This is the default type if the `type` property is not defined.
- **number**: Represents a column that binds to numeric data. It supports formatting options for displaying numbers.
- **boolean**: Represents a column that binds to boolean data. It displays checkboxes for boolean values.
- **date**: Represents a column that binds to date data. It supports formatting options for displaying dates.
- **datetime**: Represents a column that binds to date and time data. It supports formatting options for displaying date and time values.
- **checkbox**: Represents a column that displays checkboxes.

Here is an example of how to specify column types in a gantt using the types mentioned above.

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule, SelectionService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit, NgModule } from '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ SelectionService ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [splitterSettings] = "splitterSettings" [taskFields]="taskSettings"
    [treeColumnIndex]='1'>`
})
```



```

        <e-columns>
            <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=90 type='number'></e-column>
            <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=270 type='string'></e-column>
            <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=150 format='yMd' type='date' ></e-column>
            <e-column field='EndDate' headerText='End Date'
textAlign='Right' width=150 format='dd/MM/yyyy hh:mm' type='dateTime' ></e-
column>
            <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90 type='number'></e-column>
            <e-column field='Progress' headerText='Progress'
textAlign='Right' width=120 type='number'></e-column>
            <e-column field='Verified' headerText='Verified' width=100
type='boolean' displayAsCheckBox='true'></e-column>
        </e-columns>
    </ejs-gantt>,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public ngOnInit(): void {
        this.data = GanttData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks',
            Verified: 'Verified'
        };
        this.splitterSettings = {
            position: '75%'
        };
    }
}

```

### 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 `type` is not defined, then it will be determined from the first record of the [dataSource](#).

\* In case if the first record of the [dataSource](#) is null/blank value for a column then it is necessary to define the **type** for that column. This is because the gantt uses the column type to determine which filter dialog to display for that column.

#### *Difference between boolean type and checkbox type column*

1. Use the **boolean** column type when you want to bind boolean values from your datasource and/or edit boolean property values from your type.
2. Use the **checkbox** column type when you want to enable selection/deselection of the whole row.
3. When the gantt column **type** is a **checkbox**, the selection type of the gantt [selectionSettings](#) will be multiple. This is the default behavior.
4. If you have more than one column with the column type as a **checkbox**, the gantt will automatically enable the other column's checkbox when selecting one column checkbox.

To learn more about how to render boolean values as checkboxes in a Syncfusion GanttColumn, please refer to the [Render Boolean Values as Checkbox](#) section.

#### Column width

To adjust the column width in a Gantt, you can use the [width](#) property within the [column](#) property of the Gantt configuration. This property enables you to define the column width in pixels or as a percentage. You can set the width to a specific value, like **100** for 100 pixels, or as a percentage value, such as **25%** for 25% of the available width.

1. Gantt column width is calculated based on the sum of column widths. For example, a gantt container with 4 columns and a width of 800 pixels will have columns with a default width of 200 pixels each.
2. If you specify widths for some columns but not others, the Gantt will distribute the available width equally among the columns without explicit widths. For example, if you have 3 columns with widths of 100px, 200px, and no width specified for the third column, the third column will occupy the remaining width after accounting for the first two columns.
3. Columns with percentage widths are responsive and adjust their width based on the size of the gantt container. For example, a column with a width of 50% will occupy 50% of the gantt width and will adjust proportionally when the gantt container is resized to a smaller size.
4. When you manually resize columns in Syncfusion Gantt, a minimum width is set to ensure that the content within the cells remains readable. By default, the minimum width is set to 10 pixels if not explicitly specified for a column.
5. If the total width of all columns exceeds the width of the gantt container, a horizontal scrollbar will automatically appear to allow horizontal scrolling within the gantt.
6. If the parent element has a fixed width, the gantt component will inherit that width and occupy the available space. However, if the parent element does not have a fixed width, the gantt component will adjust its width dynamically based on the available space.

To learn more about resizing, you can refer to the resizing section [here](#)

#### Supported types for column width:

Syncfusion Gantt supports the following three types of column width:

##### 1. Auto

The column width is automatically calculated based on the content within the column cells. If the content exceeds the width of the column, it will be truncated with an ellipsis (...) at the end. You can set the width for columns as **auto** in your Gantt configuration as shown below:

```
`html
```

```
<e-column field='TaskID' headerText='Task ID' textAlign='Right' width='auto'></e-column>
```

•

## 2. Percentage

The column width is specified as a percentage value relative to the width of the gantt container. For example, a column width of 25% will occupy 25% of the total gantt width. You can set the width for columns as **percentage** in your Gantt configuration as shown below:

```
`html
```

```
<e-column field='TaskID' headerText='Task ID' textAlign='Right' width='25%'></e-column>
```

1

### 3. Pixel

The column width is specified as an absolute pixel value. For example, a column width of 100px will have a fixed width of 100 pixels regardless of the gantt container size. You can set the width for columns as **pixel** in your Gantt configuration as shown below:

```
`html
```

```
<e-column field='TaskID' headerText='Task ID' textAlign='Right' width='100'></e-column>
```

—

## APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
'@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[splitterSettings] = "splitterSettings" [taskFields]="taskSettings"
[treeColumnIndex]="1">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Right' width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=270 ></e-column>
        <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=120 ></e-column>
        <e-column field='Duration' headerText='Duration'
textAlign='Right' width='auto'></e-column>
    `
})
export class AppComponent {
  data: GanttData;
  taskSettings: TaskSettings;
  splitterSettings: SplitterSettings;
  ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = TaskSettings;
    this.splitterSettings = SplitterSettings;
  }
}
```

```

        <e-column field='Progress' headerText='Progress'
textAlign='Right' width='30%'></e-column>
    </e-columns>
</ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public ngOnInit(): void {
        this.data = GanttData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        };
        this.splitterSettings = {
            position: '90%'
        };
    }
}

```

### MAIN.TS

```

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

```

### Column formatting

Column formatting is a powerful feature in Syncfusion Gantt that allows you to customize the display of data in gantt columns. You can apply different formatting options to columns based on your requirements, such as displaying numbers with specific formats, formatting dates according to a specific locale, and using templates to format column values.

You can use the [columns.format](#) property to specify the format for column values.

### APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,
    selector: 'app-root',

```

```

template:
  `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[splitterSettings] = "splitterSettings" [taskFields]="taskSettings"
[treeColumnNameIndex]='1'>
    <e-columns>
      <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90 ></e-column>
      <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=290 ></e-column>
      <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=120 format= 'yyyy/MM/dd'></e-column>
      <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90></e-column>
      <e-column field='Progress' headerText='Progress'
textAlign='Right' width=120 format= 'C2' type='number'></e-column>
    </e-columns>
  </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
}

```

## MAIN.TS

```

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

```

- \* The Gantt uses the [Internalization](#) library to format values based on the specified format and culture.
- \* By default, the [number](#) and [date](#) values are formatted in **en-US** locale. You can localize the currency and date in different locale as explained [here](#).
- \* The available format codes may vary depending on the data type of the column.

\* You can also customize the formatting further by providing a custom function to the [format](#) property, instead of a format string.

\* Make sure that the format string is valid and compatible with the data type of the column, to avoid unexpected results.

### Number formatting

Number formatting allows you to customize the display of numeric values in Gantt columns. You can use standard numeric format strings or custom numeric format strings to specify the desired format. The [columns.format](#) property can be used to specify the number format for numeric columns. For example, you can use the following format strings to format numbers:

Format | Description | Remarks

{ type:'date', format:'dd/MM/yyyy' } | 04/07/1996

{ type:'date', format:'dd.MM.yyyy' } | 04.07.1996

{ type:'date', skeleton:'short' } | 7/4/96

{ type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' } | 04/07/1996 12:00 AM

{ type: 'dateTime', format: 'MM/dd/yyyy hh:mm:ss a' } | 07/04/1996 12:00:00 AM

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

```

export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public formatOptions?: object;
  public formatOptionstwo?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.formatOptions = { type: 'date', format: 'dd/MM/yyyy' };
    this.formatOptionstwo = { type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' };
    this.splitterSettings = {
      position: '75%'
    };
  }
}

```

### MAIN.TS

```

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

```

To learn more about date formatting, you can refer to [Date formatting](#).

#### *Format the date column based on localization*

You can also format the date column based on the localization settings of the user's browser. You can use the [format](#) property of the Gantt columns along with the [locale](#) property to specify the desired date format based on the locale.

In this example, the format property specifies the date format as "yyyy-MMM-dd", and the locale property specifies the locale as "es-AR" for Spanish (Argentina).

### APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { setCulture, loadCldr, setCurrencyCode } from '@syncfusion/ej2-base';
import cagregorian from './ca-gregorian.json';
import currencies from './currencies.json';
import numbers from './numbers.json';
import timeZoneNames from './timeZoneNames.json';

```

```

import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `

```



```
}

```

### MAIN.TS

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

```

#### Format template column value

Template columns in Gantt provide a way to customize the appearance of column values using HTML templates. In addition to HTML markup, you can also use number formatting to format the value displayed in a template column. To format values in a column template, you can use Angular pipes and the [format](#) property. In this example, we are using the date pipe to format the **StartDate** value as a date in the format **'dd/MMM/yyyy'**.

```
`ts

```

```
<e-column field='StartDate' headerText='Start Date' textAlign='Left' width=120>

```

```
<ng-template #template let-data>

```

```
{% raw %}

```

```
{{ data.StartDate | date:'dd/MMM/yyyy' }}

```

```
{% endraw %}

```

```
</ng-template>

```

```
</e-column>

```

```
`

```

### APP.COMPONENT.TS

```
{% raw %}
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { GanttData } from './data';
@Component({
  selector: 'app-root',
  template:
    `

```

```
export class AppComponent{
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks',
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
}
{% enddraw %}
```

### 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 other Angular pipes, such as **currency**, **decimal**, **percent**, etc., to format other types of values in the column template based on your requirements.

#### Custom formatting

Syncfusion Gantt allows you to customize the formatting of data in the gantt columns. You can apply custom formats to numeric or date columns to display data in a specific way according to the requirements. To apply custom formatting to gantt columns in Syncfusion Gantt, you can use the [format](#) property. Here's an example of how you can use custom formatting for numeric and date columns:

In the below example, the **numberFormatOptions** object is used as the **format** property for the **'Progress'** column to apply a custom numeric format with four decimal places. Similarly, the **dateFormatOptions** object is used as the **format** property for the **'StartDate'** column to apply a custom date format displaying the date in the format of day-of-the-week, month abbreviation, day, and 2-digit year (e.g. Sun, May 8, '23).

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[splitterSettings] = "splitterSettings" [taskFields]="taskSettings"
[treeColumnIndex]='1'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name' textAlign='Left'
width=270 ></e-column>
        <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=170 [format]='dateFormatOptions'></e-column>
        <e-column field='Duration' headerText='Duration' textAlign='Right'
width=90></e-column>
        <e-column field='Progress' headerText='Progress' textAlign='Right'
width=120 [format]='numberFormatOptions'></e-column>
      </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public numberFormatOptions?: object;
  public dateFormatOptions?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.dateFormatOptions = {
      // Custom format for date columns
      type: 'date',
      format: "EEE, MMM d, 'yy",
    };
    this.numberFormatOptions = {
      // Custom format for numeric columns
      format: '##.0000',
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
}

```

**MAIN.TS**

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

To learn more about custom formatting, you can refer to [Custom Date formatting](#) and [Custom Number formatting](#).

### Align the text of content

You can align the text in the content of a Gantt column using the [textAlign](#) property. This property allows you to specify the alignment of the text within the cells of a particular column in the Gantt. By default, the text is aligned to the left, but you can change the alignment by setting the value of the [textAlign](#) property to one of the following options:

*Left:* Aligns the text to the left (default). **Center:** Aligns the text to the center. *Right:* Aligns the text to the right. **Justify:** Align the text to the justify.

Here is an example of using the [textAlign](#) property to align the text of a Gantt column:

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule, DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<div style="display: flex">
      <label style="padding: 30px 17px 0 0;">Align the text for
columns :</label>
      <ejs-dropdownlist style="padding: 26px 0 0 0" index="0"
width="100" [dataSource]="alignmentData" (change)
="changeAlignment($event)"></ejs-dropdownlist>
    </div>
    <ejs-gantt id="ganttDefault" #gantt height="430px"
[dataSource]="data" [splitterSettings]="splitterSettings"
[treeColumnIndex]='1' [taskFields]="taskSettings">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name' textAlign='Left'
width=270 ></e-column>
        <e-column field='StartDate' headerText='Start Date'
textAlign='Right' format='yMd' width=150 ></e-column>
        <e-column field='Duration' headerText='Duration' textAlign='Right'
width=90></e-column>
```

```

        <e-column field='Progress' headerText='Progress' textAlign='Right'
width=120></e-column>
    </e-columns>
</ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    @ViewChild('gantt')
    public gantt?: GanttComponent;
    public alignmentData: Object[] = [
        { text: 'Left', value: 'Left' },
        { text: 'Right', value: 'Right' },
        { text: 'Center', value: 'Center' },
        { text: 'Justify', value: 'Justify' },
    ];
    public changeAlignment(args: ChangeEventArgs): void {
        (this.gantt as GanttComponent).treeGrid.grid.columns.forEach((col: any)
=> {
            col.textAlign = args.value;
        });
        (this.gantt as GanttComponent).treeGrid.refresh();
    }
    public ngOnInit(): void {
        this.data = GanttData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        };
        this.splitterSettings = {
            position: '75%'
        };
    }
}

```

### MAIN.TS

```

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

```

\* The `textAlign` property changes the alignment for both the column content and header. If you want to align header differently, you can use the `headerTextAlign` property.

### Render boolean value as checkbox

The Gantt component allows you to render boolean values as checkboxes in column. This can be achieved by using the [displayAsCheckBox](#) property, which is available in the [column](#). This property is useful when you have a boolean column in your Gantt and you want to display the values as checkboxes instead of the default text representation of **true** or **false**.

To enable the rendering of boolean values as checkboxes, you need to set the `displayAsCheckBox` property of the `columns` to **true**.

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
 '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [splitterSettings] = "splitterSettings"
    [treeColumnIndex]='1'>
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
        width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name'
        textAlign='Left' width=270 ></e-column>
        <e-column field='verified' headerText= 'Verified'
        displayAsCheckBox= true type= 'boolean' textAlign='Center'></e-column>
        <e-column field='StartDate' headerText='Start Date'
        textAlign='Right' width=120 ></e-column>
        <e-column field='Duration' headerText='Duration'
        textAlign='Right' width=90></e-column>
        <e-column field='Progress' headerText='Progress'
        textAlign='Right' width=120></e-column>
      </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
```

```

        child: 'subtasks',
        verified: 'verified'
    };
    this.splitterSettings = {
        position: '75%'
    };
}
}

```

## MAIN.TS

```

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

```

\* The `displayAsCheckBox` property is only applicable to boolean values in Gantt columns.

\* When `displayAsCheckBox` is set to **true**, the boolean values will be rendered as checkboxes in the Gantt column, with checked state indicating **true** and unchecked state indicating **false**.

### *How to prevent checkbox for particular row*

To prevent the checkbox in the particular row of the Gantt, even if the `displayAsCheckBox` property is set to true for that column, you can use the `rowDataBound` event and check for index value in the row data. If the row index and required index position matches, you can set the inner HTML of the corresponding cell to an empty string to hide the checkbox.

Here is an example of how you can prevent a checkbox from being displayed in a particular row in a Gantt:

## APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { RowDataBoundEventArgs } from '@syncfusion/ej2-angular-grids';
import { GanttData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,
    selector: 'app-root',
    template:
        `<ejs-gantt id="gantDefault" #gantt height="430px" [dataSource]="data"
        [taskFields]="taskSettings" [treeColumnIndex]='1' [splitterSettings] =
        "splitterSettings" (rowDataBound)='rowDataBound($event)' >
            <e-columns>
                <e-column field='TaskID' headerText='Task ID' textAlign='Right'
                width=90 ></e-column>
                <e-column field='TaskName' headerText='Task Name'
                textAlign='Left' width=270 ></e-column>
            </e-columns>
        `
})

```

```

        <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90></e-column>
        <e-column field='verified' headerText= 'Verified'
displayAsCheckBox= true type= 'boolean' textAlign='Center'></e-column>
        <e-column field='Progress' headerText='Progress'
textAlign='Right' width=120></e-column>
    </e-columns>
</ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: object[];
    @ViewChild('ganttt')
    public gantt?: GanttComponent;
    public taskSettings?: object;
    public splitterSettings?: object;
    public ngOnInit(): void {
        this.data = GanttData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks',
            verified: 'verified'
        };
        this.splitterSettings = {
            position: '75%'
        };
    }
    rowDataBound(args: RowDataBoundEventArgs) {
        if (args.row?.ariaRowIndex === '3') {
            args.row.children[3].innerHTML = '';
        }
    }
}

```

## MAIN.TS

```

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

```

## AutoFit columns

The Gantt has a feature that allows to automatically adjust column widths based on the maximum content width of each column when you double-click on the resizer symbol located in a specific column header. This feature ensures that all data in the gantt rows is displayed without wrapping. To use this feature, you need to inject the **ResizeService** in the provider section of **AppModule** and enable the resizer symbol in the column header by setting the [allowResizing](#) property to true in the gantt.

The following screenshot represents the resizing the column using resizer symbol.



ID	Job Name	Start Date	End Date	Mar 31, 2019
1	Project initiation	4/2/2019	4/5/2019	
2	Identify site location	4/2/2019	4/2/2019	
3	Perform Soil test	4/2/2019	4/5/2019	
4	Soil test approval	4/5/2019	4/5/2019	
5	Project estimation	4/8/2019	4/15/2019	
6	Develop floor plan for estimation	4/8/2019	4/10/2019	
7	List materials	4/11/2019	4/15/2019	
8	Estimation approval	4/15/2019	4/15/2019	
9	Sign contract	4/16/2019	4/16/2019	
10	Project approval and kick off	4/16/2019	4/16/2019	

#### *Resizing a column to fit its content using method support*

The `autoFitColumns` method of treegrid object in gantt instance resizes the column to fit the widest cell's content without wrapping. You can autofit specific columns at initial rendering by invoking the `autoFitColumns` method in `dataBound` event.

To use `autoFitColumns` method, you need to inject `ResizeService` in the provider section of `AppModule`.

#### **APP.COMPONENT.TS**

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule, ResizeService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule,
  ],
  providers: [ResizeService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
    [splitterSettings] = "splitterSettings" [allowResizing] = "true"
    (dataBound)='dataBound()'`>
    <e-columns>
      <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90></e-column>
      <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=270></e-column>
      <e-column field='StartDate' headerText='Start Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  `
})
export class AppComponent {
  taskSettings: any;
  splitterSettings: any;
  dataBound(): void {
    this.gantt.autoFitColumns();
  }
}
```

```

        <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90></e-column>
        <e-column field='Progress' headerText='Progress'
textAlign='Right' width=120></e-column>
    </e-columns>
    </ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    // Data for Gantt
    public data?: object[];
    @ViewChild('ganttt')
    public gantt?: Gantt
    public taskSettings?: object;
    public splitterSettings?: object;
    public ngOnInit(): void {
      this.data = GanttData;
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      };
      this.splitterSettings = {
        position: '75%'
      };
    }
    dataBound() {
      (this.gantt as Gantt).treeGrid.autoFitColumns(['TaskName']);
    }
  }
}

```

## 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 autofit all the columns by invoking the `autoFitColumns` method without specifying column names.

### Locked columns

The Syncfusion Gantt allows you to lock columns, which prevents them from being reordered and moves them to the first position. This functionality can be achieved by setting the `column.lockColumn` property of `treegrid` object in `ganttt` instance to `true`, which locks the column and moves it to the first position in the `ganttt`.

Here's an example of how you can use the `lockColumn` property to lock a column in the Syncfusion Gantt. Additionally, in the code snippet below, differentiate the locked column using CSS with a custom attributes property.

**APP.COMPONENT.TS**

```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule, ReorderService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ReorderService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt height="430px"
    [columns]='columns' [dataSource]="data" [taskFields]="taskSettings"
    [splitterSettings] = "splitterSettings" [allowReordering]='true'>
    </ejs-gantt>`,
  styleUrls: ['./app.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  @ViewChild('gantt')
  public gantt?: Gantt;
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public customAttributes?: Object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks',
    };
    this.columns = [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '270',
lockColumn: true, customAttributes: { class: 'customcss' } },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    this.splitterSettings = {
      position: '75%'
    };
    this.customAttributes = { class: 'customcss' };
  }
}

```

**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 columns**

The Syncfusion Gantt control allows you to show or hide columns dynamically by using property or methods available in the gantt. This feature can be useful when you want to customize the visibility of columns in the Gantt based on the requirements.

**Using property**

You can show or hide columns in the Angular Gantt using the [visible](#) property of each column. By setting the `visible` property to **true** or **false**, you can control whether the column should be visible or hidden in the gantt. Here's an example of how to show or hide a column in the Angular Gantt using the visible property:

In the below example, the **Duration** column is defined with `visible` property set to **false**, which will hide the column in the rendered gantt.

**APP.COMPONENT.TS**

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [splitterSettings] = "splitterSettings" [treeColumnIndex]='1'
    [taskFields]="taskSettings">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
        width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name' textAlign='Left'
        width=270 ></e-column>
        <e-column field='StartDate' headerText='Start Date'
        textAlign='Right' width=120 ></e-column>
        <e-column field='Duration' headerText='Duration' textAlign='Right'
        width=90 [visible]='false'></e-column>
        <e-column field='Progress' headerText='Progress' textAlign='Right'
        width=150></e-column>
      </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
```

```
// Data for Gantt
public data?: object[];
public taskSettings?: object;
public splitterSettings?: object;
public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        position: '75%'
    };
}
```

## MAIN.TS

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

\* Hiding a column using the **visible** property only affects the UI representation of the gantt. The data for the hidden column will still be available in the underlying data source, and can be accessed or modified programmatically.

\* When a column is hidden, its width is not included in the calculation of the total gantt width.

\* To hide a column permanently, you can set its visible property to false in the column definition, or remove the column definition altogether.

### Using methods

You can also show or hide columns in the Angular Gantt using the [showColumn](#) and [hideColumn](#) methods of the gantt component. These methods allow you to show or hide columns based on either the **headerText** or the **field** of the column.

### Based on header text

You can dynamically show or hide columns in the Gantt based on the header text by invoking the **showColumn** or **hideColumn** methods. These methods take either an array of column header texts or a simple string value for a column header text as the first parameter, and the value **headerText** as the second parameter to specify that you are showing or hiding columns based on the header text.

Here's an example of how to show or hide a column based on the HeaderText in the gantt:

## APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
```

```

import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
'@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id='show' (click)='show()'>Show</button>
    <button ej-button id='hide' (click)='hide()'>Hide</button>
    <br><br><br>
    <ejs-gantt id="ganttDefault" #gantt height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
    [splitterSettings] = "splitterSettings">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=270 ></e-column>
        <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90></e-column>
        <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=120 ></e-column>
        <e-column field='Progress' headerText='Progress'
textAlign='Right' width=120></e-column>
      </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  @ViewChild('gantt')
  public gantt?: GanttComponent;
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
  show(): void {
    this.gantt!.showColumn('Duration', 'headerText'); // show by
HeaderText
  }
}

```

```
hide(): void {
    this.gantt!.hideColumn('Duration', 'headerText'); // hide by
    HeaderText
}
}
```

### MAIN.TS

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

### Based on field

You can dynamically show or hide columns in the Gantt using external buttons based on the field by invoking the `showColumn` or `hideColumn` methods. These methods take either an array of column header texts or a simple string value for a column header text as the first parameter, and the value `field` as the second parameter to specify that you are showing or hiding columns based on the field.

Here's an example of how to show or hide a column based on the field in the Angular Gantt:

### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
'@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { GanttData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,
    selector: 'app-root',
    template:
        `<button ej-button id='show' (click)='show()'>Show</button>
        <button ej-button id='hide' (click)='hide()'>Hide</button>
        <br><br><br>
        <ejs-gantt id="ganttDefault" #gantt height="430px"
        [dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
        [splitterSettings] = "splitterSettings">
            <e-columns>
                <e-column field='TaskID' headerText='Task ID' textAlign='Right'
                width=90 ></e-column>
                <e-column field='TaskName' headerText='Task Name'
                textAlign='Left' width=270 ></e-column>
                <e-column field='Duration' headerText='Duration'
                textAlign='Right' width=90></e-column>
                <e-column field='StartDate' headerText='Start Date'
                textAlign='Right' width=120 ></e-column>
                <e-column field='Progress' headerText='Progress'
                textAlign='Right' width=120></e-column>
            </e-columns>
        </ejs-gantt>`,
```

```

    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    // Data for Gantt
    public data?: object[];
    @ViewChild('gantt')
    public gantt?: GanttComponent;
    public taskSettings?: object;
    public splitterSettings?: object;
    public ngOnInit(): void {
      this.data = GanttData;
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      };
      this.splitterSettings = {
        position: '75%'
      };
    }
    show(): void {
      this.gantt!.showColumn(['TaskName', 'Duration'], 'field'); // show
      by field
    }
    hide(): void {
      this.gantt!.hideColumn(['TaskName', 'Duration'], 'field'); // hide
      by field
    }
  }
}

```

### **MAIN.TS**

```

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

```

### Controlling Gantt actions

You can control various actions such as filtering, sorting, resizing, reordering, editing, and searching for specific columns in the Syncfusion Angular Gantt using the following properties:

- [allowEditing](#): Enables or disables editing for a column.
- [allowFiltering](#): Enables or disables filtering for a column.
- [allowSorting](#): Enables or disables sorting for a column.
- [allowReordering](#): Enables or disables reordering for a column.
- [allowResizing](#): Enables or disables resizing for a column.

Here is an example code that demonstrates how to control gantt actions for specific columns:

### **APP.COMPONENT.TS**



```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule, ResizeService, ReorderService, SortService,
FilterService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
'@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ResizeService, ReorderService, SortService, FilterService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt height="430px"
[dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
[splitterSettings] = "splitterSettings" [allowSorting]='true'
[allowFiltering]='true' [allowReordering]='true'
[allowResizing]='true'>
    <e-columns>
      <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=120 [allowSorting]="false"></e-column>
      <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=270 [allowFiltering]='false'></e-column>
      <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=150 ></e-column>
      <e-column field='Duration' headerText='Duration'
textAlign='Right' width=150></e-column>
      <e-column field='Progress' headerText='Progress'
textAlign='Right' width=150></e-column>
    </e-columns>
  </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  @ViewChild('gantt')
  public gantt?: Gantt
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks',
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
}

```

**MAIN.TS**

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

### Customize column styles

Customizing the gantt column styles allows you to modify the appearance of columns in the Gantt component to meet your design requirements. You can customize the font, background color, and other styles of the columns. To customize the columns styles in the gantt, you can use gantt event, css, property or method support.

For more information check on this [documentation](#).

### Manipulating columns

The Syncfusion Gantt for Angular provides powerful features for manipulating columns. This section explains how to access columns, update column definitions, and add/remove columns using Syncfusion Tree Gantt properties, methods, and events.

#### Accessing columns

To access columns in the Syncfusion Gantt component, you can use the following methods of gantt and treegrid object in gantt instance:

- [getGanttColumns:](#)

This method returns the array of columns defined in the gantt.

```
`ts
```

```
let columns = this.gantt.getGanttColumns();
```

```
`
```

- [getColumnByField:](#)

This method returns the column object that matches the specified field name.

```
`ts
```

```
let column = this.gantt.treeGrid.getColumnByField('TaskName');
```

```
`
```

- [getColumnByUid:](#)

This method returns the column object that matches the specified UID.

```
`ts
```

```
let column = this.gantt.treeGrid.getColumnByUid();
```

```
`
```

- [getVisibleColumns:](#)

This method returns the array of visible columns.

```
`ts
```

```
let visibleColumns = this.gantt.treeGrid.getVisibleColumns();
```

```
,
```

- [getColumnFieldNames](#)

This method returns an array of field names of all the columns in the Gantt.

```
`ts
```

```
let fieldNames = this.gantt.treeGrid.getColumnFieldNames()
```

```
,
```

For a complete list of column properties, refer to this [section](#)

#### *Updating column definitions*

You can update the column definitions in the Gantt using the [columns](#) property. You can modify the properties of the column objects in the columns array to update the columns dynamically. For example, you can change the headerText, width, visible, and other properties of a column to update its appearance and behavior in the gantt and then call the [refreshColumns](#) method of treegrid object in gantt instance to apply the changes to the gantt.

#### **APP.COMPONENT.TS**

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id="btnId" cssClass="e-info"
    (click)="updateColumns()"> Update Columns </button>
    <ejs-gantt id="ganttDefault" #gantt height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
    [splitterSettings] = "splitterSettings">
      <e-columns>
        <e-column field='TaskID' headerText='Task ID' textAlign='Right'
        width=90 ></e-column>
        <e-column field='TaskName' headerText='Task Name'
        textAlign='Left' width=270 ></e-column>
        <e-column field='StartDate' headerText='Start Date'
        textAlign='Right' width=120 ></e-column>
```

```

        <e-column field='Duration' headerText='Duration'
        textAlign='Right' width=90></e-column>
        <e-column field='Progress' headerText='Progress'
        textAlign='Right' width=120></e-column>
    </e-columns>
    </ejs-gantt>`,
    styleUrls: ['./app.component.css'],
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    // Data for Gantt
    public data?: object[];
    @ViewChild('ganttt')
    public gantt?: Gantt;
    public taskSettings?: object;
    public splitterSettings?: object;
    public customAttributes?: object;
    public ngOnInit(): void {
      this.data = GanttData;
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
      };
      this.splitterSettings = {
        position: '75%'
      };
    }
    updateColumns(): void {
      // Modifying column properties
      var column: any = (this.gantt as Gantt).treeGrid.columns;
      column[0].textAlign = 'Center';
      column[0].width = '100';
      column[2].visible = false;
      column[1].customAttributes = {
        class: 'customcss',
      };
      // Applying changes to the gantt
      (this.gantt as Gantt).treeGrid.refreshColumns();
    }
  }
}

```

## MAIN.TS

```

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

```

### Adding/removing Columns

The Gantt component allows you to dynamically add or remove columns to and from the gantt using the [columns](#) property, which can be accessed through the instance of the Gantt.

To add a new column to the Gantt Columns, you can directly **push** the new column object to the columns property. To remove a column from the Gantt Columns, you can use the **pop** method, which removes the last element from the columns array of the Gantt Columns. Alternatively, you can use the splice method to remove a specific column from the columns array.

Here's an example of how you can add and remove a column from the gantt:

#### APP.COMPONENT.TS

```
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='add' cssClass="e-info" (click)='addColumnns()'>
Add Column</button>
    <button ej2-button id='delete' cssClass="e-info"
(click)='deleteColumns()'> Delete Column</button>
    <ejs-gantt id="ganttDefault" #gantt height="430px"
[dataSource]="data" [taskFields]="taskSettings" [treeColumnIndex]='1'
[splitterSettings] = "splitterSettings">
    <e-columns>
    <e-column field='TaskID' headerText='Task ID' textAlign='Right'
width=90 ></e-column>
    <e-column field='TaskName' headerText='Task Name' textAlign='Left'
width=270 ></e-column>
    <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=120 ></e-column>
    <e-column field='Duration' headerText='Duration' textAlign='Right'
width=90></e-column>
    <e-column field='Progress' headerText='Progress' textAlign='Right'
width=150></e-column>
    </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  @ViewChild('gantt')
  public gantt?: GanttComponent
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
```

```

        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    };
    this.splitterSettings = {
        position: '75%'
    };
}
addColumnns(): void {
    var newColumns = [
        { field: 'TaskID', headerText: 'TaskID', width: 120 },
        { field: 'StartDate', headerText: 'StartDate', width: 120, format:
'yMd' },
    ];
    newColumns.forEach((col) => {
        (this.gantt as GanttComponent).treeGrid.grid.columns.push(col as any);
    });
    (this.gantt as GanttComponent).treeGrid.grid.refreshColumns();
}
deleteColumns(): void {
    (this.gantt as GanttComponent).treeGrid.grid.columns.pop();
    (this.gantt as GanttComponent).treeGrid.grid.refreshColumns();
}
}

```

## 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 refresh columns

You can use the [refreshColumns](#) method of treegrid object in gantt instance to refresh the columns in the gantt. This method can be used when you need to update the gantt columns dynamically based on user actions or data changes.

```
`ts
```

```
this.gantt.treeGrid.refreshColumns();
```

```
,
```

### Responsive columns

The Syncfusion Angular Gantt provides a built-in feature to toggle the visibility of columns based on media queries using the [hideAtMedia](#) property of the column object. The `hideAtMedia` accepts valid [Media Queries](#).

In this example, we have a gantt that displays data with two columns: **Task Name and Duration**. We have set the `hideAtMedia` property of the **Task Name** column to (min-width: 700px) which means that this column will be hidden when the browser screen width is less than or equal to 700px.

## APP.COMPONENT.TS

```

import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, ViewChild, OnInit, NgModule } from
 '@angular/core';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `// column hides when browser screen width lessthan 700px;
        <e-column field='TaskName' headerText='Task Name'
textAlign='Left' width=290 hideAtMedia='(min-width: 700px)'></e-column>
        <e-column field='StartDate' headerText='Start Date'
textAlign='Right' width=120 ></e-column>
        // column shows when browser screen width lessthan or equalto
500px;
        <e-column field='Duration' headerText='Duration'
textAlign='Right' width=90 hideAtMedia='(max-width: 500px)'></e-column>
        <e-column field='Progress' headerText='Progress'
textAlign='Right' width=150></e-column>
      </e-columns>
    </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.splitterSettings = {
      position: '75%'
    };
  }
}

```

**MAIN.TS**

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

## Timeline in Angular Gantt component

In the Gantt component, timeline is used to represent the project duration as individual cells with defined unit and formats.

### Timeline view modes

Gantt contains the following in-built timeline view modes:

- Hour – Minute
- Day – Hour
- Week – Day
- Month – Week
- Year – Month

Timescale mode in the Gantt component can be defined using the [timelineViewMode](#) property, and you can define a timescale mode for the top tier and bottom tier using the [topTier.unit](#) and [bottomTier.unit](#) properties.

### Week timeline mode

In the **Week** timeline mode, the upper part of the schedule header displays the weeks, whereas the bottom half of the header displays the days. Refer to the following code example.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
```



```

        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 4, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 4, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
this.timelineSettings = {
    timelineViewMode: 'Week',
};
}
}

```

### MAIN.TS

```

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

```

### Month timeline mode

In the **Month** timeline mode, the upper part of the schedule header displays the months, whereas the bottom header of the schedule displays its corresponding weeks. Refer to the following code example.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 14, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 14, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 14, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 14, Progress: 50
          },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 14, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 14, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',

```

```

        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.timelineSettings = {
        timelineViewMode: 'Month',
        timelineUnitSize: 150,
    };
}
}

```

### MAIN.TS

```

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

```

### Year timeline mode

In the **Year** timeline mode, the upper schedule header displays the years whereas, the bottom header displays its corresponding months. Refer to the following code example.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),

```

```

        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 54, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 54, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 54, Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 54, Progress: 50
},
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 54, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 54, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
this.timelineSettings = {
    timelineViewMode: 'Year',
    timelineUnitSize: 70
};
}
}

```

## MAIN.TS

```

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

```

### Day timeline mode

In the **Day** timeline mode, the upper part of the header displays the days whereas, the bottom schedule header displays its corresponding hours. Refer to the following code example.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 2, Progress: 50,isParent:false },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 2, Progress: 50, resources: [2, 3,
5],isParent:false },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 2,Predecessor:"2FS", Progress:
50,isParent:false },
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.timelineSettings = {
      timelineViewMode:'Day',
    };
  }
}

```

**MAIN.TS**

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

*Hour timeline mode*

An **Hour** timeline mode tracks the tasks in minutes scale. In this mode, the upper schedule header displays hour scale and the lower schedule header displays its corresponding minutes.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent: true,
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 2, Progress: 50, isParent: false },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 2, Progress: 50, resources: [2, 3,
            5], isParent: false },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 2, Predecessor: "2FS", Progress:
            50, isParent: false },
        ]
      },
    ],
  },
}
```

```

    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.timelineSettings = {
      timelineViewMode: 'Hour',
    };
  }
}

```

### MAIN.TS

```

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

```

### Week start day customization

In the Gantt component, you can customize the week start day using the [weekStartDay](#) property. By default, the [weekStartDay](#) is set to 0, which specifies the Sunday as a start day of the week. But, you can customize the week start day by using the following code example.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public ngOnInit(): void {
    this.data = [

```

```

        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.timelineSettings= {
        timelineViewMode: 'Week',
        weekStartDay: 3
    };
}
}

```

**MAIN.TS**

```

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

```



### Customize automatic timescale update action

In the Gantt component, the schedule timeline will be automatically updated when the tasks date values are updated beyond the project start date and end date ranges. This can be enabled or disabled using the [updateTimescaleView](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [timelineSettings]="timelineSettings"
    [editSettings]="editSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
```

```

        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
    },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.timelineSettings = {
        updateTimescaleView: false
    };
    this.editSettings = {
        allowEditing: true,
        allowTaskbarEditing: true
    };
}
}

```

### MAIN.TS

```

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

```

### Timeline cells tooltip

In the Gantt component, you can enable or disable the mouse hover tooltip of timeline cells using the [timelineSettings.showTooltip](#) property. The default value of this property is `true`. The following code example shows how to enable the timeline cells tooltip in Gantt.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,
    selector: 'app-root',
    template:

```

```

`<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [columns]="columns"
[timelineSettings]="timelineSettings"></ejs-gantt>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public columns?: object[];
    public timelineSettings?: object;
    public ngOnInit(): void {
        this.data = editingData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        };
        this.columns = [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' },
        ];
        this.timelineSettings = {
            showTooltip: true
        };
    }
}

```

## MAIN.TS

```

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

```

## Splitter in Angular Gantt component

### Splitter

In the Gantt component, the Splitter separates the TreeGrid section from the Chart section. You can change the position of the Splitter when loading the Gantt component using the [splitterSettings](#) property. By splitting the TreeGrid from the chart, the width of the TreeGrid and chart sections will vary in the component. The [splitterSettings.position](#) property denotes the percentage of the TreeGrid section's width to be rendered and this property supports both pixels and percentage values. You can define the splitter position as column index value using the [splitterSettings.columnIndex](#) property. You can also define the splitter position with built-in splitter view modes by using the [splitterSettings.view](#) property. The following list is the possible values for this property:

- **Default:** Shows Grid side and Gantt side.
- **Grid:** Shows Grid side alone in Gantt.
- **Chart:** Shows chart side alone in Gantt.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [splitterSettings]="splitterSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.splitterSettings = {
      position: "50%"
    }
  }
}
```

### MAIN.TS

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

### Change splitter position dynamically

In Gantt, we can change the splitter position dynamically by using [setSplitterPosition](#) method. Either We can change the splitter position with splitter position or columnIndex values by passing these values as arguments to [setSplitterPosition](#) method. The following code example shows how to use this methods.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DropDownListComponent, DropDownListAllModule } from
 '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { ChangeEventArgs, DropDownListComponent } from '@syncfusion/ej2-
angular-dropdowns';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='changebyposition' (click)='change()'>Change
By Position</button>
    <br><br>
    <button ej2-button id='changebyindex' (click)='changei()'>Change By
Index</button>
    <br>
    <div style="padding-top: 7px; display: inline-block">Splitter
View<ejs-dropdownlist (change)='onChange($event)' [dataSource]='dropData'
value='Default' [fields]='fields'></ejs-dropdownlist></div>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public dropData?: Object[];
  public fields?: Object;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent| any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    }
  }
}
```

```

    };
    this.dropData = [
      { id: 'Default', mode: 'Default' },
      { id: 'Grid', mode: 'Grid' },
      { id: 'Chart', mode: 'Chart' },
    ];
    this.fields = { text: 'mode', value: 'id' };
  }
  change(): void {
    this.ganttObj.setSplitterPosition('50%', 'position');
  };
  changei(): void {
    this.ganttObj.setSplitterPosition(1, 'columnIndex');
  };
  onChange(e: ChangeEventArgs): any {
    let viewType: any = <string>e.value;
    this.ganttObj.setSplitterPosition(viewType, 'view');
  };
}

```

### MAIN.TS

```

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

```

### Task scheduling in Angular Gantt component

The Gantt provides support for automatic and manual task scheduling modes. It is used to indicate whether the start date and end date of all the tasks will be automatically validated or not. [taskMode](#) is the property used to change the schedule mode of a task.

The Gantt control supports three types of mode. They are:

- **Auto**: All the tasks are automatically validate.
- **Manual**: All the tasks are manually validate by the user.
- **Custom**: Both Auto and Manual tasks are render by mapped from data source.

Note: The default value of [taskMode](#) is **Auto**.

#### Automatically scheduled Tasks

When the [taskMode](#) property is set as **Auto**, the start date and end date of all the tasks in the project will be automatically validated. That is, dates are validated based on various factors such as working time, holidays, weekends and predecessors.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';

```

```

import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
    taskMode="Auto" treeColumnIndex= "1" [taskFields]="taskSettings"
    [editSettings]="editSettings" [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public toolbar?: string[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',

```

```

        duration: 'Duration',
        progress: 'Progress',
        endDate: 'EndDate',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll', 'Search'];
    this.editSettings = {
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    };
}
}

```

### MAIN.TS

```

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

```

### Manually scheduled Tasks

When the [taskMode](#) property is set as **Manual**, the start date and end date of all the tasks in the project will be same as given in the data source. That is, dates are not validated based on various factors such as dependencies between tasks, holidays, weekends, working time.

We can restrict this mode in predecessor validation alone. That is, we can automatically validate the dates based on predecessor values by enabling the [validateManualTasksOnLinking](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
`<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
taskMode="Manual" treeColumnIndex= "1" validateManualTasksOnLinking=
"true"[taskFields]="taskSettings" [editSettings]="editSettings"
[toolbar]="toolbar"></ejs-gantt>`,

```



```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        // Data for Gantt
        public data?: object[];
        public taskSettings?: object;
        public editSettings?: object;
        public toolbar?: string[];
        public ngOnInit(): void {
            this.data = [
                {
                    TaskID: 1,
                    TaskName: 'Project Initiation',
                    StartDate: new Date('04/02/2019'),
                    EndDate: new Date('04/21/2019'),
                    subtasks: [
                        { TaskID: 2, TaskName: 'Identify Site location',
                            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
                            new Date('04/02/2019'), Duration: 4, Progress: 50 },
                        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
                            new Date('04/02/2019'), Duration: 4, Progress: 50 },
                    ]
                },
                {
                    TaskID: 5,
                    TaskName: 'Project Estimation',
                    StartDate: new Date('04/02/2019'),
                    EndDate: new Date('04/21/2019'),
                    subtasks: [
                        { TaskID: 6, TaskName: 'Develop floor plan for
                            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                        { TaskID: 7, TaskName: 'List materials', StartDate: new
                            Date('04/04/2019'), Duration: 3, Progress: 50 },
                        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
                            new Date('04/04/2019'), Duration: 3, Progress: 50 }
                    ]
                },
            ];
            this.taskSettings = {
                id: 'TaskID',
                name: 'TaskName',
                startDate: 'StartDate',
                duration: 'Duration',
                progress: 'Progress',
                endDate: 'EndDate',
                dependency: 'Predecessor',
                child: 'subtasks'
            };
            this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
                'ExpandAll', 'CollapseAll', 'Search'];
            this.editSettings = {
                allowEditing: true,
                allowDeleting: true,
                allowTaskbarEditing: true,
                showDeleteConfirmDialog: true
            };
        }
    }

```

```
}
}
```

## MAIN.TS

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

## Custom

When the [taskMode](#) property is set as Custom, the scheduling mode for each tasks will be mapped from the data source field. The Boolean property [taskFields.manual](#) is used to map the manual scheduling mode field from the data source.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

```

        'isManual': true,
        'Children': [
            {
                'TaskID': 2, 'TaskName': 'Child Task 1',
                'StartDate': new Date('02/27/2017'),
                'EndDate': new Date('03/03/2017'), 'Progress': '40'
            },
            {
                'TaskID': 3, 'TaskName': 'Child Task 2',
                'StartDate': new Date('02/26/2017'),
                'EndDate': new Date('03/03/2017'), 'Progress': '40',
                'isManual': true
            },
            {
                'TaskID': 4, 'TaskName': 'Child Task 3',
                'StartDate': new Date('02/27/2017'),
                'EndDate': new Date('03/03/2017'), 'Duration': 5,
                'Progress': '40',
            }
        ]
    },
    {
        'TaskID': 5,
        'TaskName': 'Parent Task 2',
        'StartDate': new Date('03/05/2017'),
        'EndDate': new Date('03/09/2017'),
        'Progress': '40',
        'isManual': true,
        'Children': [
            {
                'TaskID': 6, 'TaskName': 'Child Task 1',
                'StartDate': new Date('03/06/2017'),
                'EndDate': new Date('03/09/2017'), 'Progress': '40'
            },
            {
                'TaskID': 7, 'TaskName': 'Child Task 2',
                'StartDate': new Date('03/06/2017'),
                'EndDate': new Date('03/09/2017'), 'Progress': '40',
            },
            {
                'TaskID': 8, 'TaskName': 'Child Task 3',
                'StartDate': new Date('02/28/2017'),
                'EndDate': new Date('03/05/2017'), 'Progress': '40',
                'isManual': true
            },
            {
                'TaskID': 9, 'TaskName': 'Child Task 4',
                'StartDate': new Date('03/04/2017'),
                'EndDate': new Date('03/09/2017'), 'Progress': '40',
                'isManual': true
            }
        ]
    },
    {
        'TaskID': 10,
        'TaskName': 'Parent Task 3',
        'StartDate': new Date('03/13/2017'),
    }
]

```

```

        'EndDate': new Date('03/17/2017'),
        'Progress': '40',
        'Children': [
            {
                'TaskID': 11, 'TaskName': 'Child Task 1',
                'StartDate': new Date('03/13/2017'),
                'EndDate': new Date('03/17/2017'), 'Progress': '40'
            },
            {
                'TaskID': 12, 'TaskName': 'Child Task 2',
                'StartDate': new Date('03/13/2017'),
                'EndDate': new Date('03/17/2017'), 'Progress': '40',
            },
            {
                'TaskID': 13, 'TaskName': 'Child Task 3',
                'StartDate': new Date('03/13/2017'),
                'EndDate': new Date('03/17/2017'), 'Progress': '40',
            },
            {
                'TaskID': 14, 'TaskName': 'Child Task 4',
                'StartDate': new Date('03/12/2017'),
                'EndDate': new Date('03/17/2017'), 'Progress': '40',
                'isManual': true
            },
            {
                'TaskID': 15, 'TaskName': 'Child Task 5',
                'StartDate': new Date('03/13/2017'),
                'EndDate': new Date('03/17/2017'), 'Progress': '40'
            }
        ]
    };
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        endDate: 'EndDate',
        dependency: 'Predecessor',
        child: 'Children',
        manual: 'isManual'
    };
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll', 'Search'];
    this.editSettings = {
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    };
    this.columns = [
        { field: 'TaskID', visible: false },
        { field: 'TaskName' },
        { field: 'isManual' }
    ];
}

```

```
}
```

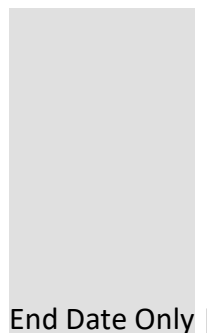
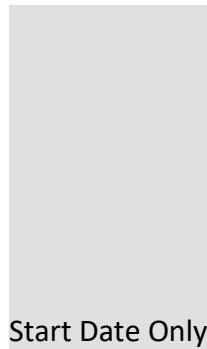
### MAIN.TS

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

### Unscheduled Tasks

Unscheduled tasks are planned for a project without any definite schedule dates. The Gantt control supports rendering the unscheduled tasks. You can create or update the tasks with anyone of start date, end date, and duration values or none. You can enable or disable the unscheduled tasks by using the [allowUnscheduledTasks](#) property. The following images represent the various types of unscheduled tasks in Gantt.

Taskbar state | Auto | Manual





Note: A milestone is a task that has no start and end dates, but it has a duration value of zero

[Define unscheduled tasks in data source](#)

You can define the various types of unscheduled tasks in the data source as follows

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    allowUnscheduledTasks='true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
```

```

        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location', Duration:
3, Progress: 50},
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', EndDate: new
Date('04/08/2019'), Progress: 50 },
        ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), EndDate: new Date('04/08/2019'),
Progress: 50, resources: [4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval',Duration:
0,Progress: 50}
        ]
    },
];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate:'EndDate',
        duration:'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.editSettings = {
        allowEditing:true,
        allowTaskbarEditing:true
    };
}
}

```

### MAIN.TS

```

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

```

### NOTE

If the [allowUnscheduledTasks](#) property is set to false, then the Gantt control automatically calculates the scheduled date values with a default value of duration 1 and the project start date is considered as the start date for the task.

### Working time range

In the Gantt control, working hours in a day for a project can be defined by using the [dayWorkingTime](#) property. Based on the working hours, automatic date scheduling and duration validations for a task are performed.

The following code snippet explains how to define the working time range for the project in Gantt.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DayMarkersService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [DayMarkersService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" highlightWeekends='true'
    [timelineSettings]="timelineSettings" [dayWorkingTime]="dayWorkingTime"
    [splitterSettings]="splitterSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public editSettings?: EditSettingsModel;
  public dayWorkingTime?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.timelineSettings = {
      timelineViewMode: 'Day'
    };
  }
}
```



```

        this.dayWorkingTime = [
            {from: 9,
             to: 18 }
        ];
        this.splitterSettings = {
            columnIndex:0
        }
    }
}

```

## MAIN.TS

```

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

```

## NOTE

- \* Individual tasks can lie between any time within the defined working time range of the project.
- \* The [dayWorkingTime](#) property is used to define the working time for the whole project.

## Weekend/non-working days

Non-working days/weekend are used to represent the non-productive days in a project. You can define the non-working days in a week using the [workWeek](#) property in Gantt.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DayMarkersService, EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [DayMarkersService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="gantDefault" height="430px" [dataSource]="data"
    highlightWeekends='true' [taskFields]="taskSettings"
    [workWeek]="workWeek"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public workWeek?: object;
  public ngOnInit(): void {

```

```

    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.workWeek =
["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday"];
  }
}

```

### 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, Saturdays and Sundays are considered as non-working days/weekend in a project.

In the Gantt control, you can make weekend as working day by setting the [includeWeekend](#) property to `true`.

### Duration units

In Gantt, the task's duration value can be measured by the following duration units,

- Day
- Hour
- Minute

In Gantt, we can define duration unit for whole project by using [durationUnit](#) property, when we defines the value for this property, this unit will be applied for all task which don't has duration unit value. And each task in the project can be defined with different duration units and the duration unit of a task can be defined by the following ways,

- Using [taskFields.durationUnit](#) property, to map the duration unit data source field.
- Defining the duration unit value along with the duration field in the data source.

### Mapping the duration unit field

The below code snippet explains the mapping of duration unit data source field to the Gantt control using the [taskFields.durationUnit](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';

```

```
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [splitterSettings] = "splitterSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      durationUnit: 'DurationUnit',
      child: 'subtasks'
    };
    this.splitterSettings = {
      columnIndex: 4
    }
  }
}
```

## 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 default value of the [durationUnit](#) property is `day`.

### *Defining duration unit along with duration field*

Duration units for the tasks can also be defined along with the duration values, the below code snippet explains the duration unit for a task along with duration value,

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
```

```

import { Gantt } from '@syncfusion/ej2-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [splitterSettings]="splitterSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent: true,
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: '3days', DurationUnit: 'day', Progress:
50, isParent: false },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: '12hours', DurationUnit: 'hour', Progress: 70,
resources: [2, 3, 5], isParent: false },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: '1800minutes', DurationUnit: 'minute',
Predecessor: "2FS", Progress: 80, isParent: false },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent: true,
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: '12hours', DurationUnit: 'hour',
Progress: 50, resources: [4], isParent: false },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: '3days', Progress: 50, DurationUnit: 'day',
resources: [4, 8], isParent: false },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: '480minutes', Predecessor: "6SS",
DurationUnit: 'minute', Progress: 70, resources: [12, 5], isParent: false }
        ]
      }
    ]
  }
}

```

```

    },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      durationUnit: 'DurationUnit',
      child: 'subtasks'
    };
    this.splitterSettings = {
      columnIndex: 4
    }
  }
}

```

### 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 edit type of the duration column in Gantt is string, to support editing the duration field along with duration units.

### Taskbar in Angular Gantt component

#### Taskbar customization

#### Taskbar height

The height of child taskbars and parent taskbars can be customized by using [taskbarHeight](#) property. The following code example shows how to use the [taskbarHeight](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskFields" [taskbarHeight]="50" [rowHeight]="60"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{

```

```

// Data for Gantt
public data?: object[];
public taskFields?: object;
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            isParent: true,
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50, isParent: false },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, resources: [2, 3,
5], isParent: false },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: "2FS", Progress:
50, isParent: false },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            isParent: true,
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4], isParent: false },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, resources: [4,
8], isParent: false },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0, Predecessor: "6SS", Progress: 50, resources:
[12, 5], isParent: false }
            ]
        },
    ];
    this.taskFields = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
}
}

```

**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 [taskbarHeight](#) value should be lower than [rowHeight](#) property value and these properties accept only pixel values.

*Conditional formatting*

The default taskbar UI can be replaced with custom templates by using the [queryTaskbarInfo](#) event. The following code example shows customizing the taskbar UI based on task progress values in Gantt component.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskFields" (queryTaskbarInfo) =
    "queryTaskbarInfo($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskFields?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
            5],isParent:false },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
            80,isParent:false },
        ]
      },
    ],
```

```

    {
      TaskID: 5,
      TaskName: 'Project Estimation',
      StartDate: new Date('04/02/2019'),
      EndDate: new Date('04/21/2019'),
      isParent:true,
      subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
          StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
            [4],isParent:false },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
          Date('04/04/2019'), Duration: 3, Progress: 70, resources: [4,
            8],isParent:false },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
          Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50, resources:
            [12, 5],isParent:false }
      ]
    },
  ];
  this.taskFields = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
  };
}
public queryTaskbarInfo(args: any) {
  if (args.data.Progress == 50) {
    args.progressBarBgColor = "red";
  } else if (args.data.Progress == 70) {
    args.progressBarBgColor = "yellow";
  } else if (args.data.Progress == 80) {
    args.progressBarBgColor = "lightgreen";
  }
};
}

```

## MAIN.TS

```

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

```

## Change gripper icon in taskbar

You can change the gripper icon in the taskbar by applying styles to their respective class elements.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';

```



```

import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings" ></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  editSettings: { allowEditing: boolean; editMode: string;
allowTaskbarEditing: boolean; } | undefined;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
    };
  }
}

```

```

        child: 'subtasks'
    };
    this.editSettings = {
        allowEditing: true,
        editMode: 'Auto',
        allowTaskbarEditing: true
    };
}
}

```

## MAIN.TS

```

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

```

## Multi Taskbar support in project view

The Gantt component, supports rendering multi-taskbars in the project view. With this feature the parent taskbar, when it is collapsed, visually summarize the progress of all its child taskbars.

This feature can be enabled by setting the [enableMultiTaskbar](#) property value to `true`.

The following code example shows how to use this property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { SelectionService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { projectViewMultiTaskData } from './data';
@Component({
    imports: [
        GanttModule
    ],
    providers: [SelectionService],
    standalone: true,
    selector: 'app-root',
    template:
        `<ejs-gantt id="ganttDefault" [dataSource]="data"
        [enableMultiTaskbar]="true" [taskFields]="taskSettings"
        [treeColumnIndex]="1" [allowSelection]="true" height="430px"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: Object[];
    public taskSettings?: object;
    public ngOnInit(): void {
        this.data = projectViewMultiTaskData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',

```

```

        startDate: 'StartDate',
        endDate: 'EndDate',
        progress: 'Progress',
        child: 'subtasks',
        expandState: 'isExpand'
    };
}
}

```

## MAIN.TS

```

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

```

## Connector lines

The width and background color of connector lines in Gantt can be customized using the [connectorLineWidth](#) and [connectorLineBackground](#) properties. The following code example shows how to use these properties.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskFields" connectorLineBackground="red"
    [connectorLineWidth]='3'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskFields?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent: true,
        subtasks: [

```

```

        { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
        { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, resources: [2, 3,
5],isParent:false },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
50,isParent:false },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    isParent:true,
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, resources: [4,
8],isParent:false },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50, resources:
[12, 5],isParent:false }
    ]
},
],
    this.taskFields = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
}
}
}

```

## MAIN.TS

```

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

```

## Enable tooltip

In the Gantt component, you can enable or disable the mouse hover tooltip for the following UI elements using the [tooltipSettings.showTooltip](#) property:

- Taskbar

- Connector line
- Baseline
- Event marker

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService, DayMarkersService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService, DayMarkersService],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

```

Date('04/08/2019'),BaselineEndDate: new Date('04/12/2019'), Duration:
4,Predecessor:"2FS", Progress: 50 },
    ]
  },
  {
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
      { TaskID: 6, TaskName: 'Develop floor plan for
estimation',BaselineStartDate: new Date('04/04/2019'),BaselineEndDate: new
Date('04/08/2019'), StartDate: new Date('04/04/2019'), Duration: 3,
Progress: 50 },
      { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),BaselineStartDate: new
Date('04/02/2019'),BaselineEndDate: new Date('04/04/2019'), Duration: 3,
Progress: 50 },
      { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/02/2019'),BaselineStartDate: new
Date('04/02/2019'),BaselineEndDate: new Date('04/08/2019'), Duration:
0,Predecessor:"6SS", Progress: 50 }
    ]
  },
];
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  duration: 'Duration',
  baselineStartDate: "BaselineStartDate",
  baselineEndDate: "BaselineEndDate",
  progress: 'Progress',
  dependency: 'Predecessor',
  child: 'subtasks'
};
this.tooltipSettings = {
  showTooltip:true
}
this.eventMarkers = [
  {
    day: '04/10/2019',
    label: 'Project approval and kick-off'
  }
];
}
}

```

## 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 value of the [tooltipSettings.showTooltip](#) property is `true`.

### Tooltip template

#### *Taskbar tooltip*

The default tooltip in the Gantt component can be customized using the [tooltipSettings.taskbar](#) property.

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

### **MAIN.TS**

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

### Connector line tooltip

The default connector line tooltip in the Gantt component can be customized using the [tooltipSettings.connectorLine](#) property. The following code example shows how to use the [tooltipSettings.connectorLine](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [tooltipSettings]="tooltipSettings"><ng-template
    #tooltipSettingsConnectorLine let-data><div> <ng-container> Offset :
    {{data.offsetString}}</ng-container> </div></ng-template></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  public data?: object[];
  public taskSettings?: object;
  public tooltipSettings?: object;
  columns: any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.tooltipSettings = {
      showTooltip: true
    };
  }
}
```

### MAIN.TS



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

### Baseline tooltip

A baseline tooltip can be customized using the [tooltipSettings.baseline](#) property. The following code example shows how to customize the baseline tooltip in Gantt.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { DayMarkersService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { baselineData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [DayMarkersService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [timelineSettings]="timelineSettings" [renderBaseline]="true"
    [treeColumnIndex]="1" height="450px" [projectStartDate]="projectStartDate"
    [projectEndDate]="projectEndDate" [dayWorkingTime]="dayWorkingTime"
    baselineColor='red']>
      <ng-template #tooltipSettingsBaseline let-data>
        <div> <ng-container>Baseline StartDate :
        {{data.BaselineStartDate}}</ng-container> </div>
      </ng-template></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  public data?: Object[];
  public taskSettings?: object;
  public columns?: object[];
  public timelineSettings?: object;
  public labelSettings?: object;
  public tooltipSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  public dayWorkingTime?: object[];
  public ngOnInit(): void {
    this.data = baselineData;
    this.taskSettings = {
      id: 'TaskId',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      baselineStartDate: 'BaselineStartDate',
```

```

        baselineEndDate: 'BaselineEndDate'
    };
    this.columns = [
        { field: 'TaskName', headerText: 'Service Name', width:
'250', clipMode: 'EllipsisWithTooltip' },
        { field: 'BaselineStartDate', headerText: 'Planned start
time' },
        { field: 'BaselineEndDate', headerText: 'Planned end time'
},
        { field: 'StartDate', headerText: 'Start time' },
        { field: 'EndDate', headerText: 'End time' },
    ];
    this.dayWorkingTime = [{ from: 0, to: 24 }];
    this.timelineSettings = {
        timelineUnitSize: 70,
        topTier: {
            unit: 'None',
        },
        bottomTier: {
            unit: 'Minutes',
            count: 15,
            format: 'hh:mm a'
        },
    };
    this.projectStartDate= new Date('03/05/2018 09:30:00 AM');
    this.projectEndDate= new Date('03/05/2018 7:00:00 PM');
    this.tooltipSettings = {
        showTooltip: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 Touch interaction

To perform **touch and hold** action on a element, refer to [tooltip popup](#).

### Critical path in Angular Gantt component

The critical path in a project is indicated by a single task or a series of tasks. If a task in critical path is delayed, the entire project will be delayed. A task is considered to be critical if any delay to this task would affect the project end date.

The critical path can be enabled in Gantt by using the built-in toolbar button or [enableCriticalPath](#) property.

The following code example shows how to display the critical path in the Gantt control:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { CriticalPathService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [CriticalPathService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[enableCriticalPath]=true [taskFields]="taskSettings" [editSettings] =
"editSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    },
    this.toolbar = ['CriticalPath']
  }
}

```

## MAIN.TS

```

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

```

### Customize taskbar in critical path

The taskbar in critical path can be customized by using [queryTaskbarInfo](#) event and [isCritical](#) property of row [data](#) in the event argument.

The following code example shows how to customize the critical path taskbar in the Gantt control:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { CriticalPathService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [CriticalPathService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[enableCriticalPath]=true [taskFields]="taskSettings"
(queryTaskbarInfo)="queryTaskbarInfo($event)" [editSettings] =
"editSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    },
    this.toolbar = ['CriticalPath']
  }
  public queryTaskbarInfo(args: any) {
```

```

        if ((args.data.isCritical || args.data.slack === '0 day') &&
!args.data.hasChildRecords) {
            args.taskbarBgColor = 'rgb(242, 210, 189)';
            args.progressBarBgColor = 'rgb(201, 169, 166)';
        }
    };
}

```

## 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 in Angular Gantt component

The Undo feature enables users to revert the most recent action performed in the Gantt Chart. It helps undo changes made to tasks, dependencies, or other actions within the Gantt Chart.

The Redo feature can reapply an action that was previously undone using the Undo feature. This allows users to revert their decision to undo an action.

The undo redo feature can be enabled in Gantt by using the [enableUndoRedo](#) property.

To use undo redo feature, inject the `UndoRedoService` in the provider section of `AppModule`.

### Configure the feature set for undo redo actions

By default, all the gantt features listed in the below table will be restored for undo and redo actions. However, you have the option to specify only the required actions to be restored using [undoRedoActions](#) property.

Built-in Undo Redo Items	Actions
----- -----	
Edit	Undo redo actions can be performed for edited record.
Delete	Undo redo actions can be performed for deleted record.
Add	Undo redo actions can be performed for newly added record.
ColumnReorder	Undo redo actions can be performed for reordered column.
Indent	Undo redo actions can be performed for indented record.
Outdent	Undo redo actions can be performed for outdented record.
ColumnResize	Undo redo actions can be performed for resized column.
Sorting	Undo redo actions can be performed for sorted column.
Filtering	Undo redo actions can be performed for filtered record.
Search	Undo redo actions can be performed for searched value.
ZoomIn	Undo redo actions can be performed for zoomIn action.
ZoomOut	Undo redo actions can be performed for zoomOut action.

- | ZoomToFit | Undo redo actions can be performed for zoomToFit action.
- | ColumnState | Undo redo actions can be performed for hided or shown columns.
- | RowDragAndDrop | Undo redo actions can be performed for row drag and drop.
- | TaskbarDragAndDrop | Undo redo actions can be performed for taskbar drag and drop.
- | PreviousTimeSpan | Undo redo actions can be performed for previous time span acton.
- | NextTimeSpan | Undo redo actions can be performed for next time span action.

In the following code example, **Edit** and **Delete** actions are specified in **undoRedoActions** property.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
  SortService, RowDDService, FilterService, ResizeService, ReorderService,
  ToolbarService, EditService, UndoRedoService, ColumnMenuService } from
  '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { ToolbarItem, EditSettingsModel, GanttAction } from
  '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers:
  [SortService, RowDDService, FilterService, ResizeService, ReorderService,
  ToolbarService, EditService, UndoRedoService, ColumnMenuService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
  [allowSorting]='true' [allowFiltering]='true' [toolbar]="toolbar"
  [enableUndoRedo]='true' [showColumnMenu]='true'
    [allowResizing]='true' [allowReordering]='true'
  [allowRowDragAndDrop]='true' [taskFields]="taskSettings"
  [undoRedoActions]="undoRedoActions" [editSettings] = "editSettings"></ejs-
  gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public undoRedoActions?: GanttAction[];
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
```

```

        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    this.editSettings = {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Search',
'ZoomIn', 'ZoomOut', 'ZoomToFit', 'Indent', 'Outdent',
'PrevTimeSpan', 'NextTimeSpan', 'Undo', 'Redo'],
    this.undoRedoActions = ['Edit', 'Delete']
}
}

```

### MAIN.TS

```

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

```

### Configuring the Storage Step Count for Undo and Redo Actions

You can specify the number of actions to be stored for undo and redo operations using the [undoRedoStepsCount](#) property.

By default, the value of `undoRedoStepsCount` is set to 10.

When the number of actions performed exceeds the `undoRedoStepsCount`, the oldest action in the undo collection is removed, and the latest action performed is added to the collection. This ensures that the number of stored actions does not exceed the specified limit, maintaining efficient memory usage.

In the following example, `undoRedoStepsCount` value is set to 5.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
SortService, RowDDService, FilterService, ResizeService, ReorderService,
ToolbarService, EditService, UndoRedoService, ColumnMenuService } from
'@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { ToolbarItem, EditSettingsModel, GanttAction } from
'@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],

```

```

providers:
[SortService, RowDDService, FilterService, ResizeService, ReorderService,
ToolbarService, EditService, UndoRedoService, ColumnMenuService],
standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[allowSorting]='true' [allowFiltering]='true' [toolbar]="toolbar"
[enableUndoRedo]='true' [showColumnMenu]='true'
  [allowResizing]='true' [allowReordering]='true'
[allowRowDragAndDrop]='true' [undoRedoStepsCount]="undoRedoStepsCount"
[taskFields]="taskSettings" [undoRedoActions]="undoRedoActions"
[editSettings] = "editSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
)})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public undoRedoStepsCount?: number;
  public undoRedoActions?: GanttAction[];
  public ngOnInit(): void {
    this.data = projectNewData;
    this.undoRedoStepsCount = 5;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    },
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Search',
'ZoomIn', 'ZoomOut', 'ZoomToFit', 'Indent', 'Outdent',
'PrevTimeSpan', 'NextTimeSpan', 'Undo', 'Redo'],
    this.undoRedoActions = ['Add', 'Edit', 'Delete',
'Search', 'Sorting', 'Filtering', 'ZoomIn', 'ZoomOut',
'ZoomToFit', 'Indent', 'Outdent',
'PreviousTimeSpan', 'NextTimeSpan', 'ColumnState']
  }
}

```

### MAIN.TS

```

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

```



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

### Perform undo redo actions programatically

You can perform undo and redo actions programatically using [undo](#) and [redo](#) methods.

The following code example demonstrates how to invoke the `undo` and `redo` method by clicking the external button.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
  SortService, RowDDService, FilterService, ResizeService, ReorderService,
  ToolbarService, EditService, UndoRedoService, ColumnMenuService } from
  '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
  '@angular/core';
import { ToolbarItem, EditSettingsModel, GanttAction, GanttComponent } from
  '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GanttModule
  ],
  providers:
  [SortService, RowDDService, FilterService, ResizeService, ReorderService,
  ToolbarService, EditService, UndoRedoService, ColumnMenuService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id='undo' (click)='undo()'>Undo</button>
    <button ej-button id='redo' (click)='redo()'>Redo</button>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [allowSorting]='true' [allowFiltering]='true'
    [toolbar]="toolbar" [enableUndoRedo]='true' [showColumnMenu]='true'
    [allowResizing]='true' [allowReordering]='true'
    [allowRowDragAndDrop]='true' [taskFields]="taskSettings"
    [undoRedoActions]="undoRedoActions" [editSettings] = "editSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public undoRedoActions?: GanttAction[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
```

```

        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    },
    this.editSettings = {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    },
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Search',
'ZoomIn', 'ZoomOut', 'ZoomToFit', 'Indent', 'Outdent',
'PrevTimeSpan', 'NextTimeSpan', 'Undo', 'Redo'],
    this.undoRedoActions = ['Add', 'Edit', 'Delete',
'Search', 'Sorting', 'Filtering', 'ZoomIn', 'ZoomOut',
'ZoomToFit', 'Indent', 'Outdent',
'PreviousTimeSpan', 'NextTimeSpan', 'ColumnState']
    }
    undo(): void {
        this.ganttObj.undo();
    };
    redo(): void {
        this.ganttObj.redo();
    };
}

```

### MAIN.TS

```

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

```

### Retrieve undo and redo stack collection

By default, when an undo or redo action is performed, the actions are stored in an array collection. To retrieve the undo and redo stack array collections, you can use the [getUndoActions](#) and [getRedoActions](#) methods.

The following code example demonstrates how to retrieve the undo and redo collection using method by clicking the external button.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
SortService, RowDDService, FilterService, ResizeService, ReorderService,
ToolbarService, EditService, UndoRedoService, ColumnMenuService } from
'@syncfusion/ej2-angular-gantt'

```

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { ToolbarItem, EditSettingsModel, GanttAction, GanttComponent } from
'@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GanttModule
  ],
  providers:
[SortService, RowDDService, FilterService, ResizeService, ReorderService,
ToolbarService, EditService, UndoRedoService, ColumnMenuService],
  standalone: true,
  selector: 'app-root',
  template:
`<button ej2-button id='getundocollection'
(click)='getundocollection()'>GetUndoCollection</button>
  <button ej2-button id='getredocollection'
(click)='getredocollection()'>GetRedoCollection</button>
  <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [allowSorting]='true' [allowFiltering]='true'
[toolbar]="toolbar" [enableUndoRedo]='true' [showColumnMenu]='true'
[allowResizing]='true' [allowReordering]='true'
[allowRowDragAndDrop]='true' [taskFields]="taskSettings"
[undoRedoActions]="undoRedoActions" [editSettings] = "editSettings"></ejs-
gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public undoRedoActions?: GanttAction[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    },
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    },
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Search',
'ZoomIn', 'ZoomOut', 'ZoomToFit', 'Indent', 'Outdent',

```

```

        'PrevTimeSpan', 'NextTimeSpan', 'Undo', 'Redo'],
        this.undoRedoActions = ['Add', 'Edit', 'Delete',
        'Search', 'Sorting', 'Filtering', 'ZoomIn', 'ZoomOut',
        'ZoomToFit', 'Indent', 'Outdent',
        'PreviousTimeSpan', 'NextTimeSpan', 'ColumnState']
    }
    getundocollection(): void {
        console.log(this.ganttObj.getUndoActions());
    };
    getredocollection(): void {
        console.log(this.ganttObj.getRedoActions());
    };
}

```

### MAIN.TS

```

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

```

### Clear undo and redo collection

At any point, you can clear the undo and redo collections using [clearUndoCollection](#) and [clearRedoCollection](#) methods. This allows you to reset the undo and redo stacks as needed during runtime.

The following code example demonstrates how to clear the undo and redo collection using method by clicking the external button.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
    SortService, RowDDService, FilterService, ResizeService, ReorderService,
    ToolbarService, EditService, UndoRedoService, ColumnMenuService } from
    '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
    '@angular/core';
import { ToolbarItem, EditSettingsModel, GanttAction, GanttComponent } from
    '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [
        GanttModule
    ],
    providers:
    [SortService, RowDDService, FilterService, ResizeService, ReorderService,
    ToolbarService, EditService, UndoRedoService, ColumnMenuService],
    standalone: true,
    selector: 'app-root',
    template:

```

```

        <button ejs-button id='clearundocollection'
(click)='clearundocollection()'>ClearUndoCollection</button>
        <button ejs-button id='clearredocollection'
(click)='clearredocollection()'>ClearRedoCollection</button>
        <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [allowSorting]='true' [allowFiltering]='true'
[toolbar]="toolbar" [enableUndoRedo]='true' [showColumnMenu]='true'
[allowResizing]='true' [allowReordering]='true'
[allowRowDragAndDrop]='true' [taskFields]="taskSettings"
[undoRedoActions]="undoRedoActions" [editSettings] = "editSettings"></ejs-
gantt>`,
        encapsulation: ViewEncapsulation.None
    })
}
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItem[];
    public undoRedoActions?: GanttAction[];
    @ViewChild('gantt', {static: true})
    public ganttObj?: GanttComponent| any;
    public ngOnInit(): void {
        this.data = projectNewData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        },
        this.editSettings = {
            allowAdding: true,
            allowEditing: true,
            allowDeleting: true,
            allowTaskbarEditing: true,
            showDeleteConfirmDialog: true
        },
        this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Search',
'ZoomIn', 'ZoomOut', 'ZoomToFit', 'Indent', 'Outdent',
'PrevTimeSpan', 'NextTimeSpan', 'Undo', 'Redo'],
        this.undoRedoActions = ['Add', 'Edit', 'Delete',
'Search', 'Sorting', 'Filtering', 'ZoomIn', 'ZoomOut',
'ZoomToFit', 'Indent', 'Outdent',
'PreviousTimeSpan', 'NextTimeSpan', 'ColumnState']
    }
    clearundocollection(): void {
        this.ganttObj.clearUndoCollection();
    };
    clearredocollection(): void {
        this.ganttObj.clearRedoCollection();
    };
}

```

## MAIN.TS

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

## Toolbar in Angular Gantt component

The Gantt component provides the toolbar support to handle Gantt actions. The [toolbar](#) property accepts the collection of built-in toolbar items and [ItemModel](#) objects for custom toolbar items.

To use toolbar feature, inject the [ToolbarService](#) in the provider section of [AppModule](#).

### Built-in toolbar items

Built-in toolbar items execute standard actions of the Gantt component, and these items can be added to toolbar by defining the [toolbar](#) as a collection of built-in items. It renders the button with icon and text.

The following table shows built-in toolbar items and its actions.

Built-in Toolbar Items	Actions
ExpandAll	Expands all the rows.
CollapseAll	Collapses all the rows.
Add	Adds a new record.
Edit	Edits the selected record.
Indent	Indent the selected record to one level.
Outdent	Outdent the elected record to one level.
Update	Updates the edited record.
Delete	Deletes the selected record.
Cancel	Cancels the edit state.
Search	Searches the records by the given key.
PrevTimeSpan	Navigate the Gantt timeline to previous time span.
NextTimeSpan	Navigate the Gantt timeline to Next time span.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService, ToolbarService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
import { projectNewData } from './data';
@Component({
  imports: [
```

```

    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
    'ExpandAll',
    'CollapseAll', 'PrevTimeSpan', 'NextTimeSpan', 'Indent', 'Outdent'];
  }
}

```

### MAIN.TS

```

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

```

\* The [toolbar](#) has options to define both built-in and custom toolbar items.

### Custom toolbar items

Custom toolbar items can be added to the toolbar by defining the [toolbar](#) property as a collection of `ItemModels`.

Actions for this customized toolbar items are defined in the [toolbarClick](#) event.

By default, the custom toolbar items are at left position. You can change the position by using the `align` property. In the following sample, the `Quick Filter` toolbar item is positioned at right.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService, ToolbarService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [FilterService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
    allowFiltering='true' [columns]="columns"
    (toolbarClick)="toolbarClick($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public toolbar?: any;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.columns = [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '250' },
```



```

        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    this.toolbar = [{text: 'Quick Filter', tooltipText: 'Quick Filter',
id: 'toolbarfilter', prefixIcon: 'e-quickfilter', align:'Right'}];
    }
    public toolbarClick(args: ClickEventArgs): void {
        if (args.item.id === 'toolbarfilter') {
            this.ganttObj.filterByColumn('TaskName', 'startswith',
'Identify');
        }
    };
}

```

### MAIN.TS

```

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

```

\* The [toolbar](#) has options to define both built-in and custom toolbar items.

\* If a toolbar item does not match the built-in items, it will be treated as a custom toolbar item.

#### Built-in and custom items in toolbar

The Gantt component has an option to use both built-in and custom toolbar items at the same time.

In the following example, the **ExpandAll** and **CollapseAll** are built-in toolbar items and **Test** is the custom toolbar item.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService, ToolbarService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [FilterService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [toolbar]="toolbar" [columns]="columns"
(toolbarClick)="toolbarClick($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})

```

```

}))
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public columns?: object[];
    public toolbar?: any;
    public ngOnInit(): void {
        this.data = editingData;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            endDate: 'EndDate',
            duration: 'Duration',
            progress: 'Progress',
            dependency: 'Predecessor',
            child: 'subtasks'
        };
        this.columns = [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' },
        ];
        this.toolbar = ['ExpandAll', 'CollapseAll', { text: 'Click',
tooltipText: 'Click',id: 'Click' } ];
    }
    public toolbarClick(args: ClickEventArgs): void {
        if (args.item.text === 'Click') {
            alert("Custom toolbar click...");
        }
    };
}

```

### MAIN.TS

```

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

```

### Enable/disable toolbar items

You can enable or disable the toolbar items by using the `enableItems` method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService, ToolbarService } from '@syncfusion/ej2-angular-
gantt'

```

```

import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [FilterService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id='enabletoolbar'
(click)='enable()'>Enable</button>
    <button ej-button id='disabletoolbar'
(click)='disable()'>Disable</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
(toolbarClick)="toolbarClick($event)" [allowFiltering]='true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: any;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.toolbar = [{ text: 'QuickFilter', id: 'QuickFilter' }, {
text: 'ClearFilter', id: 'ClearFilter' }];
  }
  public toolbarClick(args: ClickEventArgs): void {
    if (args.item.text === 'QuickFilter') {
      this.ganttObj.filterByColumn('TaskName', 'startswith',
'Identify');
    }
    if (args.item.text === 'ClearFilter') {
      this.ganttObj.clearFiltering();
    }
  }
};

```

```

enable(): void {
    this.ganttObj.toolbarModule.enableItems(['QuickFilter',
'ClearFilter'], true); // enable toolbar items.
}
disable(): void {
    this.ganttObj.toolbarModule.enableItems(['QuickFilter',
'ClearFilter'], false); // disable toolbar items.
}
}

```

## MAIN.TS

```

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

```

## Add input elements in toolbar

In the Gantt toolbar, you can add EJ2 editor elements like numeric text box, drop-down list, and date picker controls. The following code snippets demonstrates how to add EJ2 editors to the Gantt toolbar.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { ToolbarService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { Toolbar } from '@syncfusion/ej2-navigations';
import { NumericTextBox } from '@syncfusion/ej2-inputs';
import { ToolbarItem } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule, ToolbarModule
  ],
  providers: [ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
`
    <ejs-toolbar (created)="onCreate($event)">
    <e-items>
      <e-item template=' <input id="numeric" type="text" />' type =
'Input'></e-item>
    </e-items>
    </ejs-toolbar>
    <ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];

```

```

public taskSettings?: object;
public templateEle: any = new NumericTextBox({ format: 'c2', value: 1,
width: 150 });
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        }
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
}
public onCreate (e: any) {
    this.templateEle.appendTo('#numeric');
}
}

```

## MAIN.TS

```

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

```

## Managing tasks in Angular Gantt component

The Gantt component has options to dynamically insert, delete, and update tasks in the project. The primary key column is necessary to manage the tasks and perform CRUD operations in Gantt. To define the primary key, set the [columns.isPrimaryKey](#) property to **true** in the particular column.

To use CRUD, inject the [EditService](#) in the provider section of **AppModule**.

**Note:** If the [Edit](#) module is not injected, you cannot edit the tasks through the TreeGrid cells.

The following code example demonstrates editing in the Gantt component.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [columns]="columns"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public editSettings?: EditSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
```

```

        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
};
this.editSettings = {
    allowEditing: true,
    mode: "Auto"
};
this.columns = [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250' },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
];
}
}

```

## MAIN.TS

```

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

```

### Troubleshoot: Editing works only when primary key column is defined

Editing feature requires a primary key column for CRUD operations. While defining columns in Gantt using the [columns](#) property, it is mandatory that any one of the columns, must be a primary column. By default, the [id](#) column will be the primary key column. If [id](#) column is not defined, we need to enable [isPrimaryKey](#) for any one of the columns defined in the [columns](#) property.

### Open new task dialog with default values

You can set default values when new task dialog opens using [actionBegin](#) event when `requestType` is `beforeOpenAddDialog`.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [editSettings]="editSettings"
    [toolbar]="toolbar" (actionBegin)="onActionBegin($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public toolbar?: string[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
  }
};

```



```

    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.editSettings = {
      allowAdding: true
    };
    this.toolbar = ['Add'];
  }
  public onActionBegin(args: any) {
    if (args.requestType == 'beforeOpenAddDialog') {
      args.rowData.TaskName = 'Gantt';
      args.rowData.Progress = 70;
      args.rowData.ganttProperties.taskName = 'Gantt';
      args.rowData.ganttProperties.progress = 70;
    }
  };
}

```

## MAIN.TS

```

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

```

## Cell edit type and its params

The [columns.editType](#) is used to define the edit type for any particular column.

You can set the [columns.editType](#) based on data type of the column.

- **numericedit** - [NumericTextBox](#) component for integers, double, and decimal data types.
- **defaultedit** - [TextBox](#) component for string data type.
- **dropdownedit** - [DropDownList](#) component to show all unique values related to that field.
- **booleanedit** - [CheckBox](#) component for boolean data type.
- **datepickeredit** - [DatePicker](#) component for date data type.
- **datetimepickeredit** - [DateTimePicker](#) component for date time data type.

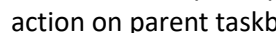
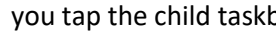
Also, you can customize the behavior of the editor component through the [columns.edit.params](#).

The following table describes cell edit type component and their corresponding edit params of the column.

Edit Type | Component | Example

[Cell editing](#) | To perform **double tap** on a specific cell, initiate the cell to be in edit state.

[Dialog editing](#) | To perform **double tap** on a specific row, initiate the edit dialog to be opened.

[Taskbar editing](#) | Taskbar editing action is initiated using the **tap** action on the taskbar. <br> **Parent taskbar** : Once you tap on the parent taskbar, it will be changed to editing state. Perform only dragging action on parent taskbar editing. <br>  <br> **Child taskbar** : Once you tap the child taskbar, it will be changed to editing state. <br>  <br> **Dragging taskbar** : To drag a taskbar to the left or right in editing state. <br> <br> **Resizing taskbar** : To resize a taskbar, drag the left/right resize icon. <br> <br> **Progress resizing** : To change the progress, drag the progress resize icon to the left or right direction.


#### [Task dependency editing](#)


You can **tap** the left/right connector point to initiate [task dependencies](#) edit mode and again tap another taskbar to establish the dependency line between two taskbars.

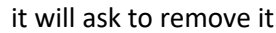
The following table explains the taskbar state in dependency edit mode.


![Taskbar states](images/taskbar-states.PNG)

Taskbar state | Description

**Parent taskbar** | You cannot create dependency relationship to parent tasks. <br> 

**Taskbar without dependency** | If you tap a valid child taskbar, it will create **FS** type dependency line between tasks, otherwise exits from task dependency edit mode. <br> 

**Taskbar with dependency** | If you tap the second taskbar, which has already been directly connected, it will ask to remove it. <br> 

**Removing dependency** | Once you tap the taskbar with direct dependency, then confirmation dialog will be shown for removing dependency. <br> 

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings"
    [editSettings]="editSettings" (load)="load()"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
```

```

// Data for Gantt
public data?: object[];
public taskSettings?: object;
public editSettings?: object;
@ViewChild('gantt', {static: true})
public ganttObj?: GanttComponent;
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 3, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 4,Predecessor:"6SS", Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.editSettings = {
        allowTaskbarEditing: true
    };
}
public load() {
    let ganttObj: any = (((document as
any).getElementById('ganttDefault'))).ej2_instances[0];
    ganttObj.isAdaptive = true; // Forcing desktop layout to change as
mobile layout
};
}

```

**MAIN.TS**

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

Note: In mobile device, you cannot create dependency other than FS by taskbar editing. By using cell/dialog editing, you can add all type of dependencies.

**Taskbar editing tooltip**

The taskbar editing tooltip can be customized using the [tooltipSettings.editing](#) property. The following code example shows how to customize the taskbar editing tooltip in Gantt.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
[editSettings]="editSettings" [tooltipSettings]="tooltipSettings">
  <ng-template #tooltipSettingsEditing let-data>
    <div> <ng-container>Duration : {{data.duration}}</ng-container>
  </div>
  </ng-template></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  public data?: object[];
  public taskSettings?: object;
  public tooltipSettings?: object;
  public editSettings?: EditSettingsModel;
  columns: any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      baselineStartDate:"BaselineStartDate",
```

```

        baselineEndDate: "BaselineEndDate",
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.editSettings = {
        allowEditing: true,
        allowTaskbarEditing: true
    },
    this.tooltipSettings = {
        showTooltip: true
    };
}
}

```

### MAIN.TS

```

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

```

### Scrolling in Angular Gantt component

The scrollbar will be displayed in the gantt when content exceeds the element **width** or **height**. The vertical and horizontal scrollbars will be displayed based on the following criteria:

- The vertical scrollbar appears when the total height of rows present in the gantt exceeds its element height.
- The horizontal scrollbar appears when the sum of columns width exceeds the grid pane size.
- The [height](#) and [width](#) are used to set the gantt height and width, respectively.

The default value for **height** and **width** is **auto**.

#### Set width and height

We can even set pixel values to width and height of gantt container using [width](#) and [height](#) properties.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:

```

```

    `<ejs-gantt id="ganttDefault" height="350px" width="600px"
[dataSource]="data" [taskFields]="taskSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public editSettings?: EditSettingsModel;
    public ngOnInit(): void {
      this.data = projectNewData;
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
      };
    }
  }
}

```

## MAIN.TS

```

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

```

## Responsive with the parent container

Specify the [width](#) and [height](#) as **100%** to make the gantt element fill its parent container.

Setting the **height** to **100%** requires the gantt parent element to have explicit height. Also, the component will be responsive when the parent container is resized.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="100%" width="100%"
[dataSource]="data" [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})

```

```

    })
    export class AppComponent{
        // Data for Gantt
        public data?: object[];
        public taskSettings?: object;
        public editSettings?: EditSettingsModel;
        public ngOnInit(): void {
            this.data = projectNewData;
            this.taskSettings = {
                id: 'TaskID',
                name: 'TaskName',
                startDate: 'StartDate',
                duration: 'Duration',
                progress: 'Progress',
                child: 'subtasks'
            };
        }
    }
}

```

### 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 in Angular Gantt component

Virtual Scroll support in Gantt allows you to load large amount of data without performance degradation. To enable Virtual Scrolling, you need to inject **VirtualScroll** module in Gantt.

### Row virtualization

Row virtualization allows you to load and render a large number of tasks in Gantt with effective performance. In this mode, all tasks are fetched initially from the datasource and rendered in the DOM within a compact viewport area.

The number of records displayed in the Gantt is determined by the height.

This mode can be enable by setting the **enableVirtualization** property to **true**.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { VirtualScrollService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent, VirtualScrollService } from '@syncfusion/ej2-angular-gantt';
@Component({
    imports: [
        GanttModule
    ],
    providers: [VirtualScrollService],
    standalone: true,

```

```

    selector: 'app-root',
    template:
      `<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
[taskFields]="taskSettings" [treeColumnIndex]="1"
    [splitterSettings]="splitterSettings" [columns]="columns"
[labelSettings]="labelSettings"
    [allowSelection]="true" [enableVirtualization]="true"
[highlightWeekends]="true"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
}
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public labelSettings?: object;
  public ngOnInit(): void {
    let tempData: any[] = [
      {
        TaskID: 1, TaskName: 'Product concept',StartDate: new
Date('04/02/2019'), EndDate: new Date('04/21/2019'),
        parentID: 0
      },
      {
        TaskID: 2, TaskName: 'Defining the product and its usage',
StartDate: new Date('04/02/2019'),
        Duration: 3, Progress: 30, parentID: 1
      },
      {
        TaskID: 3, TaskName: 'Defining target audience', StartDate: new
Date('04/02/2019'),
        parentID: 1, Duration: 3
      },
      {
        TaskID: 4, TaskName: 'Prepare product sketch and notes', StartDate:
new Date('04/05/2019'),
        Duration: 2, parentID: 1, Progress: 30
      },
      {
        TaskID: 5, TaskName: 'Concept approval', StartDate: new
Date('04/08/2019'),
        parentID: 0, Duration: 0
      },
      {
        TaskID: 6, TaskName: 'Market research', StartDate: new
Date('04/02/2019'),
        parentID: 0, EndDate: new Date('04/21/2019')
      },
      {
        TaskID: 7, TaskName: 'Demand analysis', StartDate: new
Date('04/04/2019'),
        EndDate: new Date('04/21/2019'), parentID: 6
      },
      {
        TaskID: 8, TaskName: 'Customer strength', StartDate: new
Date('04/09/2019'),

```



```

        Duration: 4, parentID: 7, Progress: 30
    },
    {
        TaskID: 9, TaskName: 'Market opportunity analysis', StartDate: new
Date('04/09/2019'),
        Duration: 4, parentID: 7
    },
    {
        TaskID: 10, TaskName: 'Competitor analysis', StartDate: new
Date('04/15/2019'),
        Duration: 4, parentID: 6, Progress: 30
    },
    {
        TaskID: 11, TaskName: 'Product strength analysis', StartDate: new
Date('04/15/2019'),
        Duration: 4, parentID: 6
    },
    {
        TaskID: 12, TaskName: 'Research complete', StartDate: new
Date('04/18/2019'),
        Duration: 0, parentID: 6
    },
    {
        TaskID: 13, TaskName: 'Product design and development', StartDate:
new Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019')
    },
    {
        TaskID: 14, TaskName: 'Functionality design', StartDate: new
Date('04/19/2019'),
        Duration: 3, parentID: 13, Progress: 30
    },
    {
        TaskID: 15, TaskName: 'Quality design', StartDate: new
Date('04/19/2019'),
        Duration: 3, parentID: 13
    },
    {
        TaskID: 16, TaskName: 'Define reliability', StartDate: new
Date('04/24/2019'),
        Duration: 2, Progress: 30, parentID: 13
    },
    {
        TaskID: 17, TaskName: 'Identifying raw materials', StartDate: new
Date('04/24/2019'),
        Duration: 2, parentID: 13
    },
    {
        TaskID: 18, TaskName: 'Define cost plan', StartDate: new
Date('04/04/2019'),
        parentID: 13, EndDate: new Date('04/21/2019')
    },
    {
        TaskID: 19, TaskName: 'Manufacturing cost', StartDate: new
Date('04/26/2019'),
        Duration: 2, Progress: 30, parentID: 18
    },
    },

```

```

    {
      TaskID: 20, TaskName: 'Selling cost', StartDate: new
Date('04/26/2019'),
      Duration: 2, parentID: 18
    },
    {
      TaskID: 21, TaskName: 'Development of the final design', StartDate:
new Date('04/30/2019'),
      parentID: 13, EndDate: new Date('04/21/2019')
    },
    {
      TaskID: 22, TaskName: 'Defining dimensions and package volume',
StartDate: new Date('04/30/2019'),
      Duration: 2, parentID: 21, Progress: 30
    },
    {
      TaskID: 23, TaskName: 'Develop design to meet industry standards',
StartDate: new Date('05/02/2019'),
      Duration: 2, parentID: 21
    },
    {
      TaskID: 24, TaskName: 'Include all the details', StartDate: new
Date('05/06/2019'),
      Duration: 3, parentID: 21
    },
    {
      TaskID: 25, TaskName: 'CAD computer-aided design', StartDate: new
Date('05/09/2019'),
      Duration: 3, parentID: 13, Progress: 30
    },
    {
      TaskID: 26, TaskName: 'CAM computer-aided manufacturing', StartDate:
new Date('09/14/2019'),
      Duration: 3, parentID: 13
    },
    {
      TaskID: 27, TaskName: 'Design complete', StartDate: new
Date('05/16/2019'),
      Duration: 0, parentID: 13
    },
    {
      TaskID: 28, TaskName: 'Prototype testing', StartDate: new
Date('05/17/2019'),
      Duration: 4, Progress: 30, parentID: 0
    },
    {
      TaskID: 29, TaskName: 'Include feedback', StartDate: new
Date('05/17/2019'),
      Duration: 4, parentID: 0
    },
    {
      TaskID: 30, TaskName: 'Manufacturing', StartDate: new
Date('05/23/2019'),
      Duration: 5, Progress: 30, parentID: 0
    },
  ],
  {

```

```

    TaskID: 31, TaskName: 'Assembling materials to finsihed goods',
    StartDate: new Date('05/30/2019'),
    Duration: 5, parentID: 0
  },
  {
    TaskID: 32, TaskName: 'Feedback and testing', StartDate: new
    Date('04/04/2019'),
    parentID: 0, EndDate: new Date('04/21/2019'),
  },
  {
    TaskID: 33, TaskName: 'Internal testing and feedback', StartDate:
    new Date('06/06/2019'),
    Duration: 3, parentID: 32, Progress: 45
  },
  {
    TaskID: 34, TaskName: 'Customer testing and feedback', StartDate:
    new Date('06/11/2019'),
    Duration: 3, parentID: 32, Progress: 50
  },
  {
    TaskID: 35, TaskName: 'Final product development', StartDate: new
    Date('04/04/2019'),
    parentID: 0, EndDate: new Date('04/21/2019'),
  },
  {
    TaskID: 36, TaskName: 'Important improvements', StartDate: new
    Date('06/14/2019'),
    Duration: 4, Progress: 30, parentID: 35
  },
  {
    TaskID: 37, TaskName: 'Address any unforeseen issues', StartDate:
    new Date('06/14/2019'),
    Duration: 4, Progress: 30, parentID: 35
  },
  {
    TaskID: 38, TaskName: 'Final product', StartDate: new
    Date('04/04/2019'),
    parentID: 0, EndDate: new Date('04/21/2019'),
  },
  {
    TaskID: 39, TaskName: 'Branding product', StartDate: new
    Date('06/20/2019'),
    Duration: 4, parentID: 38
  },
  {
    TaskID: 40, TaskName: 'Marketing and presales', StartDate: new
    Date('06/26/2019'), Duration: 4,
    Progress: 30, parentID: 38
  }
];
let virtualData: any[] = [];
let projId: number = 1;
for (let i: number = 0; i < 50; i++) {
  let x: number = virtualData.length + 1;
  let parent: any = {};
  /* tslint:disable:no-string-literal */
  parent['TaskID'] = x;

```

```

    parent['TaskName'] = 'Project ' + (i + 1);
    virtualData.push(parent);
    for (let j: number = 0; j < tempData.length; j++) {
        let subtasks: any = {};
        /* tslint:disable:no-string-literal */
        subtasks['TaskID'] = tempData[j].TaskID + x;
        subtasks['TaskName'] = tempData[j].TaskName;
        subtasks['StartDate'] = tempData[j].StartDate;
        subtasks['Duration'] = tempData[j].Duration;
        subtasks['Progress'] = tempData[j].Progress;
        subtasks['parentID'] = tempData[j].parentID + x;
        virtualData.push(subtasks);
    }
}
this.data = virtualData,
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    parentID: 'parentID'
};
this.columns = [
    { field: 'TaskID' },
    { field: 'TaskName' },
    { field: 'StartDate' },
    { field: 'Duration' },
    { field: 'Progress' }
];
this.splitterSettings = {
    columnIndex: 2
};
this.labelSettings = {
    leftLabel: 'TaskName',
    taskLabel: 'Progress'
};
}
}
}

```

### MAIN.TS

```

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

```

### Timeline virtualization

Timeline virtualization allows you to load a data source having large timespan with high performance. Initially, it renders the timeline with thrice the width of the gantt element, while other timeline cells render on-demand during horizontal scrolling.

This mode can be enable by setting the [enableTimelineVirtualization](#) property to `true`.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { VirtualScrollService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { GanttComponent, VirtualScrollService } from '@syncfusion/ej2-
angular-gantt';
import { ToolbarItem, EditSettingsModel } from '@syncfusion/ej2-angular-
gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [VirtualScrollService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="450px" [dataSource]="data"
    [taskFields]="taskSettings" [treeColumnIndex]="1"
    [splitterSettings]="splitterSettings" [columns]="columns"
    [labelSettings]="labelSettings" [projectStartDate]="projectStartDate"
    [projectEndDate]="projectEndDate"
    [allowSelection]="true" [enableVirtualization]="true"
    [enableTimelineVirtualization]="true" [autoCalculateDateScheduling]="false"
    [editSettings] = "editSettings" [highlightWeekends]="true"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItem[];
  public labelSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  public ngOnInit(): void {
    let tempData: any[] = [
      {
        TaskID: 1, TaskName: 'Product concept',StartDate: new
Date('04/02/2019'), EndDate: new Date('04/21/2019'),
        parentID: 0
      },
      {
        TaskID: 2, TaskName: 'Defining the product and its usage',
StartDate: new Date('04/02/2019'),
        Duration: 3, Progress: 30, parentID: 1
      },
      {
        TaskID: 3, TaskName: 'Defining target audience', StartDate: new
Date('04/02/2019'),
        parentID: 1, Duration: 3
      },
    ],
  }
}

```

```

    {
      TaskID: 4, TaskName: 'Prepare product sketch and notes', StartDate:
new Date('04/05/2019'),
      Duration: 2, parentID: 1, Progress: 30
    },
    {
      TaskID: 5, TaskName: 'Concept approval', StartDate: new
Date('04/08/2019'),
      parentID: 0, Duration: 0
    },
    {
      TaskID: 6, TaskName: 'Market research', StartDate: new
Date('04/02/2019'),
      parentID: 0, EndDate: new Date('04/21/2019')
    },
    {
      TaskID: 7, TaskName: 'Demand analysis', StartDate: new
Date('04/04/2019'),
      EndDate: new Date('04/21/2019'), parentID: 6
    },
    {
      TaskID: 8, TaskName: 'Customer strength', StartDate: new
Date('04/09/2019'),
      Duration: 4, parentID: 7, Progress: 30
    },
    {
      TaskID: 9, TaskName: 'Market opportunity analysis', StartDate: new
Date('04/09/2019'),
      Duration: 4, parentID: 7
    },
    {
      TaskID: 10, TaskName: 'Competitor analysis', StartDate: new
Date('04/15/2019'),
      Duration: 4, parentID: 6, Progress: 30
    },
    {
      TaskID: 11, TaskName: 'Product strength analysis', StartDate: new
Date('04/15/2019'),
      Duration: 4, parentID: 6
    },
    {
      TaskID: 12, TaskName: 'Research complete', StartDate: new
Date('04/18/2019'),
      Duration: 0, parentID: 6
    },
    {
      TaskID: 13, TaskName: 'Product design and development', StartDate:
new Date('04/04/2019'),
      parentID: 0, EndDate: new Date('04/21/2019')
    },
    {
      TaskID: 14, TaskName: 'Functionality design', StartDate: new
Date('04/19/2019'),
      Duration: 3, parentID: 13, Progress: 30
    },
    {

```

```

    TaskID: 15, TaskName: 'Quality design', StartDate: new
Date('04/19/2019'),
    Duration: 3, parentID: 13
  },
  {
    TaskID: 16, TaskName: 'Define reliability', StartDate: new
Date('04/24/2019'),
    Duration: 2, Progress: 30, parentID: 13
  },
  {
    TaskID: 17, TaskName: 'Identifying raw materials', StartDate: new
Date('04/24/2019'),
    Duration: 2, parentID: 13
  },
  {
    TaskID: 18, TaskName: 'Define cost plan', StartDate: new
Date('04/04/2019'),
    parentID: 13, EndDate: new Date('04/21/2019')
  },
  {
    TaskID: 19, TaskName: 'Manufacturing cost', StartDate: new
Date('04/26/2019'),
    Duration: 2, Progress: 30, parentID: 18
  },
  {
    TaskID: 20, TaskName: 'Selling cost', StartDate: new
Date('04/26/2019'),
    Duration: 2, parentID: 18
  },
  {
    TaskID: 21, TaskName: 'Development of the final design', StartDate:
new Date('04/30/2019'),
    parentID: 13, EndDate: new Date('04/21/2019')
  },
  {
    TaskID: 22, TaskName: 'Defining dimensions and package volume',
StartDate: new Date('04/30/2019'),
    Duration: 2, parentID: 21, Progress: 30
  },
  {
    TaskID: 23, TaskName: 'Develop design to meet industry standards',
StartDate: new Date('05/02/2019'),
    Duration: 2, parentID: 21
  },
  {
    TaskID: 24, TaskName: 'Include all the details', StartDate: new
Date('05/06/2019'),
    Duration: 3, parentID: 21
  },
  {
    TaskID: 25, TaskName: 'CAD computer-aided design', StartDate: new
Date('05/09/2019'),
    Duration: 3, parentID: 13, Progress: 30
  },
  {
    TaskID: 26, TaskName: 'CAM computer-aided manufacturing', StartDate:
new Date('09/14/2019'),

```

```

        Duration: 3, parentID: 13
    },
    {
        TaskID: 27, TaskName: 'Design complete', StartDate: new
Date('05/16/2019'),
        Duration: 0, parentID: 13
    },
    {
        TaskID: 28, TaskName: 'Prototype testing', StartDate: new
Date('05/17/2019'),
        Duration: 4, Progress: 30, parentID: 0
    },
    {
        TaskID: 29, TaskName: 'Include feedback', StartDate: new
Date('05/17/2019'),
        Duration: 4, parentID: 0
    },
    {
        TaskID: 30, TaskName: 'Manufacturing', StartDate: new
Date('05/23/2019'),
        Duration: 5, Progress: 30, parentID: 0
    },
    {
        TaskID: 31, TaskName: 'Assembling materials to finsihed goods',
StartDate: new Date('05/30/2019'),
        Duration: 5, parentID: 0
    },
    {
        TaskID: 32, TaskName: 'Feedback and testing', StartDate: new
Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
        TaskID: 33, TaskName: 'Internal testing and feedback', StartDate:
new Date('06/06/2019'),
        Duration: 3, parentID: 32, Progress: 45
    },
    {
        TaskID: 34, TaskName: 'Customer testing and feedback', StartDate:
new Date('06/11/2019'),
        Duration: 3, parentID: 32, Progress: 50
    },
    {
        TaskID: 35, TaskName: 'Final product development', StartDate: new
Date('04/04/2019'),
        parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
        TaskID: 36, TaskName: 'Important improvements', StartDate: new
Date('06/14/2019'),
        Duration: 4, Progress: 30, parentID: 35
    },
    {
        TaskID: 37, TaskName: 'Address any unforeseen issues', StartDate:
new Date('06/14/2019'),
        Duration: 4, Progress: 30, parentID: 35
    },
}

```



```

    {
      TaskID: 38, TaskName: 'Final product', StartDate: new
Date('04/04/2019'),
      parentID: 0, EndDate: new Date('04/21/2019'),
    },
    {
      TaskID: 39, TaskName: 'Branding product', StartDate: new
Date('06/20/2019'),
      Duration: 4, parentID: 38
    },
    {
      TaskID: 40, TaskName: 'Marketing and presales', StartDate: new
Date('06/26/2019'), Duration: 4,
      Progress: 30, parentID: 38
    }
  ];
let virtualData: any[] = [];
let projId: number = 1;
for (let i: number = 0; i < 50; i++) {
  let x: number = virtualData.length + 1;
  let parent: any = {};
  /* tslint:disable:no-string-literal */
  parent['TaskID'] = x;
  parent['TaskName'] = 'Project ' + (i + 1);
  virtualData.push(parent);
  for (let j: number = 0; j < tempData.length; j++) {
    let subtasks: any = {};
    /* tslint:disable:no-string-literal */
    subtasks['TaskID'] = tempData[j].TaskID + x;
    subtasks['TaskName'] = tempData[j].TaskName;
    subtasks['StartDate'] = tempData[j].StartDate;
    subtasks['Duration'] = tempData[j].Duration;
    subtasks['Progress'] = tempData[j].Progress;
    subtasks['parentID'] = tempData[j].parentID + x;
    virtualData.push(subtasks);
  }
}
this.data = virtualData,
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  endDate: 'EndDate',
  duration: 'Duration',
  progress: 'Progress',
  parentID: 'parentID'
};
this.columns = [
  { field: 'TaskID' },
  { field: 'TaskName' },
  { field: 'StartDate' },
  { field: 'Duration' },
  { field: 'Progress' }
];
this.splitterSettings = {
  columnIndex: 2
};

```

```

this.editSettings = {
  allowAdding: true,
  allowEditing: true,
  allowDeleting: true,
  allowTaskbarEditing: true,
  showDeleteConfirmDialog: true
},
this.toolbar = ['Add', 'Cancel', 'CollapseAll', 'Delete', 'Edit',
'ExpandAll', 'NextTimeSpan', 'PrevTimeSpan', 'Search', 'Update', 'Indent',
'Outdent']
this.labelSettings = {
  leftLabel: 'TaskName',
  taskLabel: 'Progress'
};
this.projectEndDate = new Date('04/01/2019');
this.projectStartDate = new Date('12/31/2025');
}
}

```

### 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 scroll

- Due to the element height limitation in browsers, the maximum number of records loaded is limited by the browser capacity.
- Cell selection will not be persisted.
- The number of records rendered will be determined by the `height` property.
- It is necessary to mention the height of the Gantt in pixels when enabling Virtual Scrolling.

### Resources in Angular Gantt component

In Gantt, the resources are represented by staff, equipment and materials etc. In Gantt control you can show or allocate the resources (human resources) for each task.

#### Resource collection

The resource collection contains details about resources that are used in the project. Resources are JSON object that contains id, name, unit and group of the resources and this collection is mapped to the Gantt control using the [resources](#) property. These resource fields are mapped to the Gantt control using the [resourceFields](#) property.

Resource fields | Description

[id](#) | This field is used to assign resources to the tasks.

[name](#) | This field is used to map the resource names. These names are displayed as one of Gantt columns and also can display as labels using the [labelSettings](#) property.

[unit](#) | It indicates the amount of work that can be done by a resource for the task in a day.

[group](#) | This field is used to group the resources and the tasks assigned to that particular resource into category.

The following code snippets shows resource collection and how it assigned to Gantt control.

```
`typescript
var projectResources= [
{ resourceid: 1, resourceName: 'Martin Tamer', resourceGroup: 'Planning Team', resourceUnit: 50},
{ resourceid: 2, resourceName: 'Rose Fuller', resourceGroup: 'Testing Team', resourceUnit: 70 },
{ resourceid: 3, resourceName: 'Margaret Buchanan', resourceGroup: 'Approval Team' },
{ resourceid: 4, resourceName: 'Fuller King', resourceGroup: 'Development Team' },
{ resourceid: 5, resourceName: 'Davolio Fuller', resourceGroup: 'Approval Team' },
{ resourceid: 6, resourceName: 'Van Jack', resourceGroup: 'Development Team', resourceUnit: 40 },
];
export default {
data: function() {
return{
resourceFields: {
id: 'resourceid', //resource Id Mapping
name: 'resourceName', //resource Name mapping
unit: 'resourceUnit', //resource Unit mapping
group: 'resourceGroup' //resource Group mapping
}
resources: projectResources //resource collection dataSource
};
},
};
`
```

### Assign resource

We can assign resources for a task at initial load, using the resource id value of the resources as a collection. This collection is mapped from the dataSource to the Gantt control using the [resourceInfo](#) property.

Resources are assigned to tasks in following ways.

### Assign resource alone

If the unit is not specified for specific resource, the amount of work done will be consider as 100% by default. In such cases, the resource unit will not be displayed in Gantt UI.

```
`typescript
```

```
{ TaskID: 2, TaskName: 'Identify site location', StartDate: new Date('04/02/2019'), Duration: 0, Progress: 50, resources: [1] }
```

#### *Assign resource with unit*

We can assign the quantity of work done by the resources for the specific task as like below code snippet.

```
`typescript
{
TaskID: 2, TaskName: 'Identify site location', StartDate: new Date('03/29/2019'), Duration: 2,
Progress: 30, resources: [{ resourceid: 1, Unit: 70 }, 6]
}
```

When resource unit is defined in resource collection, the amount of work done by that particular resource will be same for all the tasks.

The following code snippet shows how to assign the resource for each task and map to Gantt control.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [treeColumnIndex]="1"
    [projectStartDate]="projectStartDate" [projectEndDate]="projectEndDate"
    [labelSettings]="labelSettings" [resourceFields]="resourceFields"
    [columns]="columns" [resources]="resources"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public resources?: object[];
  public taskSettings?: object;
  public labelSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  resourceFields: { id: string; name: string; unit: string; } | any;
  columns: ({ field: string; visible: boolean; headerText?: undefined;
width?: undefined; } | { field: string; headerText: string; width: string;
```

```

visible?: undefined; } | { field: string; width: string; visible?:
undefined; headerText?: undefined; })[] | undefined;
    public ngOnInit(): void {
        this.data = [
            {
                TaskID: 1,
                TaskName: 'Project initiation',
                StartDate: new Date('03/29/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {
                        TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 2,
                        Progress: 30, resources: [{ resourceId: 1, Unit: 70 }, 6]
                    },
                    {
                        TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('03/29/2019'), Duration: 4,
                        resources: [2, 3, 5]
                    },
                    {
                        TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('03/29/2019'), Duration: 1,
                        resources: [8, { resourceId: 9, Unit: 50 }], Progress: 30
                    },
                ]
            },
            {
                TaskID: 5,
                TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {
                        TaskID: 6, TaskName: 'Develop floor plan for estimation',
                        StartDate: new Date('03/29/2019'),
                        Duration: 3, Progress: 30, resources: [{ resourceId: 4,
Unit: 50 }]
                    },
                    {
                        TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/01/2019'), Duration: 3,
                        resources: [4, 8]
                    },
                    {
                        TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/01/2019'),
                        Duration: 2, resources: [12, { resourceId: 5, Unit: 70 }]
                    },
                ]
            },
            {
                TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
                Progress: 30, resources: [12]
            }
        ];
        this.taskSettings = {

```

```

        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        resourceInfo: 'resources',
        child: 'subtasks'
    };
    this.labelSettings = {
        leftLabel: 'TaskName',
        rightLabel: 'resources'
    };
    this.resourceFields= {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit'
    };
    this.columns = [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Task Name', width: '180' },
        { field: 'resources', headerText: 'Resources', width: '160' },
        { field: 'Duration', width: '100' }
    ];
    this.projectStartDate= new Date('03/25/2019');
    this.projectEndDate= new Date('07/28/2019');
    this.resources = [
        { resourceId: 1, resourceName: 'Martin Tamer' },
        { resourceId: 2, resourceName: 'Rose Fuller' },
        { resourceId: 3, resourceName: 'Margaret Buchanan' },
        { resourceId: 4, resourceName: 'Fuller King' },
        { resourceId: 5, resourceName: 'Davolio Fuller' },
        { resourceId: 6, resourceName: 'Van Jack' },
        { resourceId: 7, resourceName: 'Fuller Buchanan' },
        { resourceId: 8, resourceName: 'Jack Davolio' },
        { resourceId: 9, resourceName: 'Tamer Vinet' },
        { resourceId: 10, resourceName: 'Vinet Fuller' },
        { resourceId: 11, resourceName: 'Bergs Anton' },
        { resourceId: 12, resourceName: 'Construction Supervisor' }
    ];
}
}
}

```

### MAIN.TS

```

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

```

### Add/Edit resource collection

By using cell/ dialog edit option, we can add/remove the multiple resources for a particular task. Resource Unit can be change for a each task on resource tab in edit dialog by double click on the unit cell.

The top screenshot shows the Gantt component interface. At the top, there are buttons: 'Update', 'Cancel', 'Expand all', and 'Collapse all'. Below these are two date ranges: 'Mar 24, 2019' and 'Mar 31, 2019'. The main table has columns for 'Task Name' and 'Resources'. The tasks are grouped under 'Project initiation' and 'Project estimation'. A dropdown menu is open for the task 'Identify site locat...', showing a list of resources with checkboxes: Martin Ta..., Rose Fuller, Margaret ..., Fuller King, Davolio Fu..., and Van Jack. The bottom screenshot shows the 'Task Information' dialog box. It has two tabs: 'GENERAL' and 'RESOURCES'. The 'RESOURCES' tab is active, displaying a table with columns: ID, Name, Unit, and Group. The table lists five resources assigned to the task, with checkboxes in the first column.

ID	Name	Unit	Group	
<input checked="" type="checkbox"/>	1	Martin Tamer	70	Planning Team
<input checked="" type="checkbox"/>	2	Rose Fuller	70	Testing Team
<input type="checkbox"/>	3	Margaret Buch...	100	Approval Team
<input type="checkbox"/>	4	Fuller King	100	Development T...
<input type="checkbox"/>	5	Davolio Fuller	100	Approval Team

### Resource view in Angular Gantt component

The resource breakdown view is used to visualize the tasks assigned to each resource in hierarchy manner. Resources are displayed as parents and all the tasks assigned to each resource are displayed as its child records. It can be initialized by setting the [viewType](#) property to `ResourceView`.

### Resource task

A task assigned to one or more resources are termed as resource task and it is added as child task to the respective resource. Already assigned task can also be shared or moved with other resources by adding a resource name to the task or removing resource name from the task by cell or dialog editing.

Note: Currently there is no support for unscheduled task in Resource view Gantt.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
```

```

import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [resources]="resources" [taskFields]="taskSettings"
    [resourceFields]="resourceFields" [editSettings]="editSettings"
    [columns]="columns" [toolbar]="toolbar" [labelSettings]="labelSettings"
    [splitterSettings]="splitterSettings" [allowSelection]='true'
    [allowResizing] = 'true' [highlightWeekends] = 'true' [treeColumnIndex]="1"
    [projectStartDate]="projectStartDate" [projectEndDate]="projectEndDate"
    viewType="ResourceView"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public resources?: object[];
  public taskSettings?: object;
  public labelSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  resourceFields: { id: string; name: string; unit: string; group: string;
} | undefined;
  editSettings: { allowAdding: boolean; allowEditing: boolean;
allowDeleting: boolean; allowTaskbarEditing: boolean;
showDeleteConfirmDialog: boolean; } | undefined;
  columns: ({ field: string; visible: boolean; headerText?: undefined;
width?: undefined; } | { field: string; headerText: string; width: number;
visible?: undefined; } | { field: string; headerText: string; visible?:
undefined; width?: undefined; } | { field: string; visible?: undefined;
headerText?: undefined; width?: undefined; })[] | undefined;
  toolbar: string[] | undefined;
  splitterSettings: { columnIndex: number; } | undefined;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project initiation',
        StartDate: new Date('03/29/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          {
            TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 2,
            Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
          }
        ]
      }
    ]
  }
}

```



```

    },
    {
        TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('03/29/2019'), Duration: 4,
        resources: [{resourceId: 2, resourceUnit: 70}], Progress:
30, work: 20
    },
    {
        TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('03/29/2019'), Duration: 1,
        resources: [{resourceId: 3, resourceUnit: 25}, { resourceId:
1, resourceUnit: 75 }], Progress: 30, work: 10,
    },
]
},
{
    TaskID: 5,
    TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('03/29/2019'),
            Duration: 3, Progress: 30, resources: [{ resourceId: 4,
resourceUnit: 50 }, {resourceId: 2, resourceUnit: 70}], work: 30
        },
        {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/01/2019'), Duration: 3,
            resources: [{resourceId: 6, resourceUnit: 40}], Progress:
30, work: 40
        },
        {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/01/2019'),
            Duration: 2, resources: [{ resourceId: 5, resourceUnit: 75
}], Progress: 30, work: 60,
        }
    ]
},
{
    TaskID: 9, TaskName: 'Sign contract', StartDate: new
Date('04/01/2019'), Duration: 1,
    Progress: 30,
}
];
this.resources = [
    { resourceId: 1, resourceName: 'Martin Tamer' },
    { resourceId: 2, resourceName: 'Rose Fuller' },
    { resourceId: 3, resourceName: 'Margaret Buchanan' },
    { resourceId: 4, resourceName: 'Fuller King' },
    { resourceId: 5, resourceName: 'Davolio Fuller' },
    { resourceId: 6, resourceName: 'Van Jack' },
    { resourceId: 7, resourceName: 'Fuller Buchanan' },
    { resourceId: 8, resourceName: 'Jack Davolio' },
    { resourceId: 9, resourceName: 'Tamer Vinet' },
    { resourceId: 10, resourceName: 'Vinet Fuller' },

```

```

        { resourceId: 11, resourceName: 'Bergs Anton' },
        { resourceId: 12, resourceName: 'Construction Supervisor' }
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        resourceInfo: 'resources',
        work: 'work',
        child: 'subtasks'
    };
    this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName',
        unit: 'Unit',
        group: 'resourceGroup'
    };
    this.editSettings = {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    };
    this.columns = [
        { field: 'TaskID', visible: false },
        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
        { field: 'Progress' },
        { field: 'resourceGroup', headerText: 'Group' },
        { field: 'StartDate' },
        { field: 'Duration' }
    ];
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
'ExpandAll', 'CollapseAll'];
    this.labelSettings = {
        rightLabel: 'resources'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.projectStartDate = new Date('03/25/2019');
    this.projectEndDate = new Date('07/28/2019');
}
}

```

## MAIN.TS

```

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

```

### Resource overAllocation

When a resource is assigned too much of work to complete within a day of resource's available time then it is called as overallocation.

The available working time of resources for completing the task in a day will be calculated based on the `dayWorkingTime` property and `resource unit`.

The range of overallocation dates can be highlighted by a square bracket. It can be enabled by setting the `showOverAllocation` property as `true`. The following code example demonstrates how to hide or show the over allocation by clicking the custom button.

Note: By default, the `showOverAllocation` property value is `false`.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ToolbarItem, EditSettingsModel, SelectionSettingsModel } from
 '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

```

    @ViewChild('gantt', {static: true})
    public ganttObj?: GanttComponent;
    resourceFields: { id: string; name: string; unit: string; group: string;
} | undefined;
    editSettings: { allowAdding: boolean; allowEditing: boolean;
allowDeleting: boolean; allowTaskbarEditing: boolean;
showDeleteConfirmDialog: boolean; } | undefined;
    columns: ({ field: string; visible: boolean; headerText?: undefined;
width?: undefined; } | { field: string; headerText: string; width: number;
visible?: undefined; } | { field: string; headerText: string; visible?:
undefined; width?: undefined; } | { field: string; visible?: undefined;
headerText?: undefined; width?: undefined; })[] | undefined;
    toolbar: (string | { text: string; tooltipText: string; id: string; })[]
| undefined;
    splitterSettings: any;
    public ngOnInit(): void {
        this.data = [
            {
                TaskID: 1,
                TaskName: 'Project initiation',
                StartDate: new Date('03/29/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {
                        TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
                        Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
                    },
                    {
                        TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
                        resources: [{ resourceId: 1, resourceUnit: 70 }],
                        Predecessor: 2, Progress: 30, work: 20
                    },
                    {
                        TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
                        resources: [{ resourceId: 1, resourceUnit: 25 }],
                        Predecessor: 3, Progress: 30, work: 10,
                    },
                ]
            },
            {
                TaskID: 5,
                TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {
                        TaskID: 6, TaskName: 'Develop floor plan for estimation',
                        StartDate: new Date('04/01/2019'),
                        Duration: 5, Progress: 30, resources: [{ resourceId: 2,
resourceUnit: 50 }], work: 30
                    },
                    {
                        TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4,

```

```

        resources: [{ resourceId: 2, resourceUnit: 40 }],
Predecessor: '6FS-2', Progress: 30, work: 40
    },
    {
        TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/09/2019'),
        Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
}], Predecessor: '7FS-1', Progress: 30, work: 60,
    }
]
},
{
    TaskID: 9,
    TaskName: 'Site work',
    StartDate: new Date('04/04/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        {
            TaskID: 10, TaskName: 'Install temporary power service',
StartDate: new Date('04/01/2019'), Duration: 14,
            Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
}],
        },
        {
            TaskID: 11, TaskName: 'Clear the building site', StartDate:
new Date('04/08/2019'),
            Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
[3]
        },
        {
            TaskID: 12, TaskName: 'Sign contract', StartDate: new
Date('04/12/2019'),
            Duration: 5, resources: [3], Predecessor: '11FS-5'
        },
    ]
}
];
this.resources = [
    { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
'Planning Team' },
    { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
'Testing Team' },
    { resourceId: 3, resourceName: 'Margaret Buchanan',
resourceGroup: 'Approval Team' }
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    resourceInfo: 'resources',
    work: 'work',
    child: 'subtasks'
};

```

```

    this.resourceFields = {
      id: 'resourceId',
      name: 'resourceName',
      unit: 'Unit',
      group: 'resourceGroup'
    };
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true,
      allowTaskbarEditing: true,
      showDeleteConfirmDialog: true
    };
    this.columns = [
      { field: 'TaskID', visible: false },
      { field: 'TaskName', headerText: 'Name', width: 250 },
      { field: 'work', headerText: 'Work' },
      { field: 'Progress' },
      { field: 'resourceGroup', headerText: 'Group' },
      { field: 'StartDate' },
      { field: 'Duration' }
    ];
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel',
      'ExpandAll', 'CollapseAll',
      {text: 'Show/Hide Overallocation', tooltipText: 'Show/Hide
Overallocation', id: 'showhidebar'}];
    this.labelSettings = {
      rightLabel: 'resources',
      taskLabel: 'Progress'
    }
    this.projectStartDate = new Date('03/25/2019');
    this.projectEndDate = new Date('07/28/2019');
  }
  public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'showhidebar') {
      this.ganttObj!.showOverAllocation =
this.ganttObj!.showOverAllocation ? 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));

```

## Unassigned task

A task not assigned to any one of the resource are termed as unassigned tasks. The unassigned tasks are grouped with a name as **Unassigned Task** and displayed at the bottom of Gantt data collection . It is validated at load time during Gantt record creation by default based on a task **resourceInfo** mapping property in the Gantt chart data source. If the resource is assigned to the unassigned grouped tasks, the task will be moved as child to the respective resource.

### Enable taskbar drag and drop

In Gantt, you can enable taskbar drag and drop between resources by using the [allowTaskbarDragAndDrop](#) property. This allows you to move a taskbar from one resource to another vertically, making it easier to schedule tasks and manage resources.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, EditService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ToolbarItem, EditSettingsModel, SelectionSettingsModel } from
 '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" [dataSource]="data"
    [resources]="resources" [taskFields]="taskSettings"
    [resourceFields]="resourceFields" [editSettings]="editSettings"
    [columns]="columns" [toolbar]="toolbar" [labelSettings]="labelSettings"
    [allowSelection]='true' [allowResizing] = 'true' [highlightWeekends]
    = 'true' [treeColumnIndex]="1" [allowTaskbarDragAndDrop] = 'true'
    [projectStartDate]="projectStartDate"
    [projectEndDate]="projectEndDate" viewType="ResourceView"
    [showOverAllocation] = 'true'
    [enableMultiTaskbar]= 'true' ></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public resources?: object[];
  public taskSettings?: object;
  public labelSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  editSettings: { allowAdding: boolean; allowEditing: boolean;
allowDeleting: boolean; allowTaskbarEditing: boolean;
showDeleteConfirmDialog: boolean; } | undefined;
  columns: ({ field: string; ss: any; headerText?: undefined; width?:
undefined; } | { field: string; headerText: string; width: number; ss?:
undefined; } | { field: string; headerText: string; ss?: undefined; width?:
undefined; } | { field: string; ss?: undefined; headerText?: undefined;
width?: undefined; })[] | undefined;
  toolbar: string[] | undefined;
  resourceFields: { id: string; name: string; unit: string; group: string;
} | undefined;
  public ngOnInit(): void {
```

```

    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project initiation',
        StartDate: new Date('03/29/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          {
            TaskID: 2, TaskName: 'Identify site location', StartDate:
new Date('03/29/2019'), Duration: 3,
            Progress: 30, work: 10, resources: [{ resourceId: 1,
resourceUnit: 50 }]
          },
          {
            TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/03/2019'), Duration: 4,
            resources: [{ resourceId: 1, resourceUnit: 70 }],
            Predecessor: 2, Progress: 30, work: 20
          },
          {
            TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/09/2019'), Duration: 4,
            resources: [{ resourceId: 1, resourceUnit: 25 }],
            Predecessor: 3, Progress: 30, work: 10,
          },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project estimation', StartDate: new Date('03/29/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          {
            TaskID: 6, TaskName: 'Develop floor plan for estimation',
            StartDate: new Date('04/01/2019'),
            Duration: 5, Progress: 30, resources: [{ resourceId: 2,
resourceUnit: 50 }], work: 30
          },
          {
            TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 4,
            resources: [{ resourceId: 2, resourceUnit: 40 }],
            Predecessor: '6FS-2', Progress: 30, work: 40
          },
          {
            TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/09/2019'),
            Duration: 4, resources: [{ resourceId: 2, resourceUnit: 75
}], Predecessor: '7FS-1', Progress: 30, work: 60,
          }
        ]
      },
      {
        TaskID: 9,
        TaskName: 'Site work',
        StartDate: new Date('04/04/2019'),
        EndDate: new Date('04/21/2019'),

```



```

        subtasks: [
            {
                TaskID: 10, TaskName: 'Install temporary power service',
                StartDate: new Date('04/01/2019'), Duration: 14,
                Progress: 30, resources: [{ resourceId: 3, resourceUnit: 75
            }],
        },
        {
            TaskID: 11, TaskName: 'Clear the building site', StartDate:
            new Date('04/08/2019'),
            Duration: 9, Progress: 30, Predecessor: '10FS-9', resources:
            [3]
        },
        {
            TaskID: 12, TaskName: 'Sign contract', StartDate: new
            Date('04/12/2019'),
            Duration: 5, resources: [3], Predecessor: '11FS-5'
        },
    ]
}

];
this.resources = [
    { resourceId: 1, resourceName: 'Martin Tamer', resourceGroup:
    'Planning Team', isExpand: false},
    { resourceId: 2, resourceName: 'Rose Fuller', resourceGroup:
    'Testing Team', isExpand: true},
    { resourceId: 3, resourceName: 'Margaret Buchanan',
    resourceGroup: 'Approval Team', isExpand: false }
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    resourceInfo: 'resources',
    work: 'work',
    expandState: 'isExpand',
    child: 'subtasks'
};
this.resourceFields = {
    id: 'resourceId',
    name: 'resourceName',
    unit: 'Unit',
    group: 'resourceGroup'
};
this.editSettings = {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
};
this.columns = [
    { field: 'TaskID' },

```

```

        { field: 'TaskName', headerText: 'Name', width: 250 },
        { field: 'work', headerText: 'Work' },
        { field: 'Progress' },
        { field: 'resourceGroup', headerText: 'Group' },
        { field: 'StartDate' },
        { field: 'Duration' }
    ];
    this.toolbar = ['ExpandAll', 'CollapseAll'];
    this.labelSettings = {
        rightLabel: 'resources',
        taskLabel: 'TaskName'
    };
    this.projectStartDate = new Date('03/25/2019');
    this.projectEndDate = new Date('07/28/2019');
}

```

### 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 Gantt component

Filtering allows you to view specific or related records based on filter criteria. This can be done in the Gantt Component by using the filter menu support and toolbar search support. To enable filtering in the Gantt Component, set the [allowFiltering](#) to **true**. Menu filtering support can be configured using the [filterSettings](#) property and toolbar searching can be configured using the [searchSettings](#) property.

To use the filter, inject the [FilterService](#) in the provider section of [AppModule](#).

#### Filter hierarchy modes

The Gantt supports a set of filtering modes with the [filterSettings.hierarchyMode](#) property. The following are the types of filter hierarchy modes available in the Gantt component:

- **Parent**: This is the default filter hierarchy mode in Gantt. The filtered records are displayed with its parent records. If the filtered records do not have any parent record, then only the filtered records will be displayed.
- **Child**: Displays the filtered records with its child record. If the filtered records do not have any child record, then only the filtered records will be displayed.
- **Both**: Displays the filtered records with its both parent and child records. If the filtered records do not have any parent and child records, then only the filtered records will be displayed.
- **None**: Displays only the filtered records.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { DropDownListComponent, DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns';

```

```

import { FilterService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ChangeEventArgs, DropDownListComponent } from '@syncfusion/ej2-
angular-dropdowns';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule, DropDownListAllModule
  ],
  providers: [FilterService],
  standalone: true,
  selector: 'app-root',
  template:
    `

Copyright © 2001 -2024 Syncfusion Inc.



215


```

```

    ];
    this.splitterSettings = {
        columnIndex: 3
    };
    this.fields = { text: 'mode', value: 'id' };
}
onChange(e: ChangeEventArgs): any {
    let mode: any = <string>e.value;
    this.ganttObj.filterSettings.hierarchyMode = mode;
    this.ganttObj.clearFiltering();
};
}

```

### MAIN.TS

```

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

```

### Initial filter

To apply the filter at initial rendering, set the filter to predicate object in the [filterSettings.columns](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { FilterService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
    imports: [
        GanttModule
    ],
    providers: [FilterService],
    standalone: true,
    selector: 'app-root',
    template:
        `<ejs-gantt id="ganttDefault" height="430px" [allowFiltering]='true'
        [dataSource]="data" [taskFields]="taskSettings" [splitterSettings] =
        "splitterSettings" [columns]="columns"
        [filterSettings]="filterSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public columns?: object[];
    public filterSettings?: object;
    public ngOnInit(): void {
        this.data = [

```

```

{
  TaskID: 1,
  TaskName: 'Project initiation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
    {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
    {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
  ]
},
{
  TaskID: 5,
  TaskName: 'Project estimation',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  subtasks: [
    {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
    {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
    {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]}
  ]
}];
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
};
this.columns = [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
];
this.splitterSettings = {
  columnIndex:3
};
this.filterSettings = {
  columns: [{ field: 'TaskName', matchCase: false, operator:
'startswith', predicate: 'and', value: 'Identify' },
    { field: 'TaskID', matchCase: false, operator: 'equal',
predicate: 'and', value: 2 } ]
};
}

```

```
}

```

### MAIN.TS

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

### Filter operators

The filter operator for a column can be defined in the `filterSettings.columns.operator` property.

The available operators and its supported data types are:

Operator	Description	Supported Types
startswith	Checks whether the value begins with the specified value.	String
endswith	Checks whether the value ends with the specified value.	String
contains	Checks whether the value contains the specified value.	String
equal	Checks whether the value is equal to the specified value.	String &#124; Number &#124; Boolean &#124; Date
notequal	Checks for values not equal to the specified value.	String &#124; Number &#124; Boolean &#124; Date
greaterthan	Checks whether the value is greater than the specified value.	Number &#124; Date
greaterthanorequal	Checks whether a value is greater than or equal to the specified value.	Number &#124; Date
lessthan	Checks whether the value is less than the specified value.	Number &#124; Date
lessthanorequal	Checks whether the value is less than or equal to the specified value.	Number &#124; Date

By default, the `filterSettings.columns.operator` value is equal.

### Diacritics

By default, the Gantt component ignores the diacritic characters while filtering. To include diacritic characters, set the `filterSettings.ignoreAccent` to `true`.

In the following sample, type **Project** in the `TaskName` column to filter diacritic characters.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
```

```

providers: [FilterService],
standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [allowFiltering]='true'
[dataSource]="data" [taskFields]="taskSettings" [splitterSettings] =
"splitterSettings" [columns]="columns"
[filterSettings]="filterSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public filterSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0,Progress: 50, resources: [1]},
          {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2',Progress: 50, resources:
[2, 3, 5]},
          {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'),Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
          {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'),Duration: 3, Predecessor: '6', resources: [4,
8],Progress: 50},
          {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'),Duration: 0, Predecessor: '7', resources: [12, 5]
          }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',

```

```

        child: 'subtasks'
      };
      this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' }
      ];
      this.splitterSettings = {
        columnIndex: 3
      };
      this.filterSettings = {
        ignoreAccent: true
      };
    }
  }
}

```

### MAIN.TS

```

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

```

### Filtering a specific column by method

You can filter the columns dynamically by using the [filterByColumn](#) method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { FilterService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GanttModule
  ],
  providers: [FilterService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id='filterRecord'
(click)='filter()'>Filter</button>
    <br><br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[allowFiltering]='true' [dataSource]="data" [taskFields]="taskSettings"
[splitterSettings] = "splitterSettings" [columns]="columns"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})

```



```

export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public columns?: object[];
    @ViewChild('gantt', {static: true})
    public ganttObj?: GanttComponent;
    public ngOnInit(): void {
        this.data = [
            {
                TaskID: 1,
                TaskName: 'Project initiation',
                StartDate: new Date('04/02/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0, Progress: 50, resources: [1]},
                    {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2', Progress: 50, resources:
[2, 3, 5]},
                    {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
                ]
            },
            {
                TaskID: 5,
                TaskName: 'Project estimation',
                StartDate: new Date('04/02/2019'),
                EndDate: new Date('04/21/2019'),
                subtasks: [
                    {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
                    {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Predecessor: '6', resources: [4,
8], Progress: 50},
                    {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0, Predecessor: '7', resources: [12, 5]}
                ]
            }
        ]};
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            startDate: 'StartDate',
            duration: 'Duration',
            progress: 'Progress',
            child: 'subtasks'
        };
        this.columns = [
            { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
            { field: 'TaskName', headerText: 'Task Name', width: '250' },
            { field: 'StartDate', headerText: 'Start Date', width: '150' },
            { field: 'Duration', headerText: 'Duration', width: '150' },
            { field: 'Progress', headerText: 'Progress', width: '150' },
        ];
    }
}

```

```

        this.splitterSettings = {
            columnIndex: 3
        };
    }
    filter(): void {
        this.ganttObj!.filterByColumn('TaskName', 'startswith', 'Iden', 'and');
    }
}

```

## MAIN.TS

```

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

```

## Clear filtered columns

You can clear all the filtering conditions done in the Gantt component by using the [clearFiltering](#) method.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { FilterService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [
        GanttModule
    ],
    providers: [FilterService],
    standalone: true,
    selector: 'app-root',
    template:
        `Filter</button>  
        <br><br><br>  
        <ejs-gantt #gantt id="ganttDefault" height="430px"  
[allowFiltering]='true' [dataSource]="data" [taskFields]="taskSettings"  
[splitterSettings] = "splitterSettings" [columns]="columns"  
[filterSettings]="filterSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public columns?: object[];
    public filterSettings?: object;
}

```

```

@ViewChild('gantt', {static: true})
public ganttObj?: GanttComponent;
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                {TaskID: 2, TaskName: 'Identify site location', StartDate: new
Date('04/02/2019'), Duration: 0, Progress: 50, resources: [1]},
                {TaskID: 3, TaskName: 'Perform soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Predecessor: '2', Progress: 50, resources:
[2, 3, 5]},
                {TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 0, Predecessor: '3', Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                {TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Predecessor: '4', Progress:
50, resources: [4]},
                {TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Predecessor: '6', resources: [4,
8], Progress: 50},
                {TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0, Predecessor: '7', resources: [12, 5]}
            ]
        }
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    this.splitterSettings = {
        columnIndex: 3
    };
    this.filterSettings = {
        columns: [{ field: 'TaskName', matchCase: false, operator:
'startswith', predicate: 'and', value: 'Identify' },

```

```

        { field: 'Progress', matchCase: false, operator: 'equal',
predicate: 'and', value: 50 }]
    };
}
filter(): void {
    this.ganttObj!.clearFiltering();
};
}

```

## MAIN.TS

```

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

```

## Sorting in Angular Gantt component

Sorting enables you to sort data in the ascending or descending order. To sort a column, click the column header.

To sort multiple columns, press and hold the CTRL key and click the column header. You can clear sorting of any one of the multi-sorted columns by pressing and holding the SHIFT key and clicking the specific column header.

To enable sorting in the Gantt component, set the [allowSorting](#) property to `true`. Sorting options can be configured through the [sortSettings](#) property.

To use sort, inject the [SortService](#) in the provider section of `AppModule`.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings" [columns]="columns"
[splitterSettings]="splitterSettings" [allowSorting]= 'true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;

```

```

public splitterSettings?: object;
public columns?: object[];
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
}
}

```

**MAIN.TS**

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

\* Gantt columns are sorted in the ascending order. If you click the already sorted column, the sort direction toggles.

\* To disable sorting for a particular column, set the [columns.allowSorting](#) property to `false`.

#### Sorting column on Gantt initialization

The Gantt component can be rendered with sorted columns initially, and this can be achieved by using the [sortSettings](#) property. You can add columns that are sorted initially in the [sortSettings.columns](#) collection defined with [field](#) and [direction](#) properties. The following code example shows how to add the sorted column to Gantt initialization.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';

@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [splitterSettings]="splitterSettings" [allowSorting]= 'true'
    [sortSettings]="sortSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public splitterSettings?: object;
  public sortSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
};
this.splitterSettings = {
    columnIndex: 3
};
this.columns = [
    { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
    { field: 'TaskName', headerText: 'Task Name', width: '250' },
    { field: 'StartDate', headerText: 'Start Date', width: '150' },
    { field: 'Duration', headerText: 'Duration', width: '150' },
    { field: 'Progress', headerText: 'Progress', width: '150' },
];
this.sortSettings = { columns: [{ field: 'TaskID', direction:
'Descending' }, { field: 'TaskName', direction: 'Ascending' }] };
}
}

```

**MAIN.TS**

```

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

```

### Sorting column dynamically

Columns in the Gantt component can be sorted dynamically using the [sortColumn](#) method. The following code example demonstrates how to invoke the [sortColumn](#) method by clicking the custom button.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='sortColumn' (click)='sort()'>Sort
    Column</button>
    <br><br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
    [splitterSettings]="splitterSettings" [allowSorting]= 'true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent| any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
    ],
  },
}
```



```

        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
}
sort(): void {
    this.ganttObj.sortModule.sortColumn('TaskName', "Descending",
false);
};
}

```

### MAIN.TS

```

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

```

### Clear all the sorting dynamically

In the Gantt component, you can clear all the sorted columns and return to previous position using the [clearSorting](#) public method. The following code snippet shows how to clear all the sorted columns by clicking the custom button.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='Clearsort' (click)='sort()'>Clear
Sort</button>
    <br><br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
[splitterSettings]="splitterSettings" [sortSettings]="sortSettings"
[allowSorting]= 'true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public splitterSettings?: object;
  public sortSettings?: object;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent| any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),

```

```

        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ],
    };
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    this.sortSettings = { columns: [{ field: 'TaskID', direction: 'Descending' }, { field: 'TaskName', direction: 'Ascending' }] };
    sort(): void {
        this.ganttObj.clearSorting();
    };
}

```

### MAIN.TS

```

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

```

### Sorting events

During the sort action, the Gantt component triggers two events. The [actionBegin](#) event triggers before the sort action starts, and the [actionComplete](#) event triggers after the sort action is completed.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { SortService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';

```

```

import { Gantt } from '@syncfusion/ej2-gantt';
import { SortEventArgs } from '@syncfusion/ej2-angular-grids';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [splitterSettings]="splitterSettings" [allowSorting]= 'true'
    (actionBegin)="actionBegin($event)"
    (actionComplete)="actionComplete($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',

```

```

        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
    this.splitterSettings = {
        columnIndex: 3
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
    }
    public actionBegin(args: SortEventArgs) {
        alert(args.requestType + ' ' + args.type); //custom Action
    };
    public actionComplete(args: SortEventArgs) {
        alert(args.requestType + ' ' + args.type); //custom Action
    };
}

```

### MAIN.TS

```

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

```

The `args.requestType` is the current action name. For example, for sorting the `args.requestType`, value is **sorting**.

### Sorting Custom Columns

In Gantt, you can sort custom columns of different types like string, numeric, etc., By adding the custom column in the column collection, you can perform initial sort using the `sortSettings` or you can also sort the column dynamically by a button click.

The following code snippets explains how to achieve this.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SortService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { editingData } from './data';

```

```

@Component({
  imports: [
    GanttModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej-button id='sortColumn' (click)='sort()'>Sort Custom
    Column</button>
    <br><br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
    [splitterSettings]="splitterSettings" [allowSorting]= 'true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public columns?: object[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50, /*CustomColumn:
'BCustomColumn'*/ CustomColumn: '2' },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, /*CustomColumn:
'BCustomColumn'*/ CustomColumn: '3' },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
Date('04/02/2019'), Duration: 4, Progress: 50, /*CustomColumn:
'BCustomColumn'*/ CustomColumn: '4' },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50,
/*CustomColumn: 'BCustomColumn'*/ CustomColumn: '6' },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, /*CustomColumn:
'BCustomColumn'*/ CustomColumn: '1' },
        ]
      }
    ]
  }
}

```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50, /*CustomColumn:
'BCustomColumn'*/ CustomColumn: '5' }
    ]
  },
];
this.taskSettings = {
  id: 'TaskID',
  name: 'TaskName',
  startDate: 'StartDate',
  duration: 'Duration',
  progress: 'Progress',
  child: 'subtasks'
};
this.splitterSettings = {
  columnIndex: 3
};
this.columns = [
  { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
  { field: 'TaskName', headerText: 'Task Name', width: '250' },
  { field: 'StartDate', headerText: 'Start Date', width: '150' },
  { field: 'Duration', headerText: 'Duration', width: '150' },
  { field: 'Progress', headerText: 'Progress', width: '150' },
  { field: 'CustomColumn', headerText: 'CustomColumn', width:
'150' },
];
}
sort(): void {
  this.ganttObj.sortModule.sortColumn('CustomColumn', "Ascending",
false);
};
}

```

### MAIN.TS

```

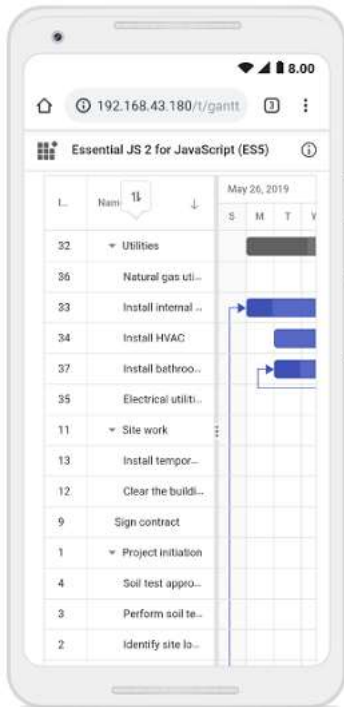
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));

```

### Touch interaction

To perform **tap** action on a column header, trigger [sorting](#) operation to the selected column. A popup is displayed for multi-column sorting. To sort multiple columns, tap the popup, and then tap the desired column headers.

The following screenshot shows Gantt touch sorting,



### Selection in Angular Gantt component

Selection provides an option to highlight a row or a cell. It can be done using arrow keys or by scrolling down the mouse. To disable selection in the Gantt component, set the [allowSelection](#) to **false**.

To select data, inject the [SelectionService](#) module in provider section of **AppModule**.

The Gantt component supports two types of selection that can be set by using the [selectionSettings.type](#) property. They are:

- **Single**: Sets a single value by default and allows only selection of a single row or a cell.
- **Multiple**: Allows you to select multiple rows or cells. To perform the multi-selection, press and hold the CTRL key and click the desired rows or cells.

### Selection mode

The Gantt component supports three types of selection modes that can be set by using the [selectionSettings.mode](#). They are:

- **Row**: Allows you to select only rows, and the row value is set by default.
- **Cell**: Allows you to select only cells.
- **Both**: Allows you to select rows and cells at the same time.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
```



```

import { Gantt } from '@syncfusion/ej2-gantt';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [selectionSettings]="selectionSettings"></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public selectionSettings?: SelectionSettingsModel;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',

```

```

        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.selectionSettings = {
        mode: 'Both',
    };
}
}

```

## MAIN.TS

```

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

```

## Toggle selection

The toggle selection allows you to select and deselect a specific row or cell. To enable toggle selection, set the `enableToggle` property of the `selectionSettings` to `true`. If you click the selected row or cell, then it will be deselected and vice versa. By default, the `enableToggle` property is set to `false`.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { SelectionService } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [
        GanttModule
    ],
    providers: [SelectionService],
    standalone: true,
    selector: 'app-root',
    template:
`<button ej2-button id='toggle' (click)='toggle()'>Disable
toggle</button>
<br><br>
<ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"
[selectionSettings]="selectionSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public selectionSettings?: SelectionSettingsModel;
}

```

```

@ViewChild('gantt', {static: true})
public ganttObj?: GanttComponent | any;
public ngOnInit(): void {
    this.data = [
        {
            TaskID: 1,
            TaskName: 'Project Initiation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
                { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            ]
        },
        {
            TaskID: 5,
            TaskName: 'Project Estimation',
            StartDate: new Date('04/02/2019'),
            EndDate: new Date('04/21/2019'),
            subtasks: [
                { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
                { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
            ]
        },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.selectionSettings = {
        mode: 'Row',
        type: 'Multiple',
        enableToggle: true
    };
    toggle(): void {
        this.ganttObj.selectionSettings.enableToggle = false;
    };
}

```

**MAIN.TS**

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

### Clear selection

You can clear the selected cells and selected rows by using a method called [clearSelection](#). The following code example demonstrates how to clear the selected rows in Gantt Chart.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='selectMultipleRow' (click)='select()'>Select
Multiple Rows</button>
    <button ej2-button id='clearselection' (click)='clear()'>Clear
Selection</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"
[selectionSettings]="selectionSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public selectionSettings?: SelectionSettingsModel;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
```

```

        { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ]
},
{
    TaskID: 5,
    TaskName: 'Project Estimation',
    StartDate: new Date('04/02/2019'),
    EndDate: new Date('04/21/2019'),
    subtasks: [
        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
},
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    dependency: 'Predecessor',
    child: 'subtasks'
};
this.selectionSettings = {
    mode: 'Row',
    type: 'Multiple',
};
}
select(): void {
    this.ganttObj.selectionModule.selectRows([1, 3, 5]); // passing the
record index to select the rows
};
clear(): void {
    this.ganttObj.clearSelection(); // Clear the selected rows
};
}

```

### MAIN.TS

```

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

```

Get selected row indexes and records

You can get the selected row indexes by using the [getSelectedRowIndex](#) method. And by using [getSelectedRecords](#) method, you can get the selected record details.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [selectionSettings]="selectionSettings"
    (rowSelected) = "rowSelected($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public selectionSettings?: SelectionSettingsModel;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
        ]
      }
    ]
  }
}

```

```

        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ],
    },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.selectionSettings = {
        mode: 'Row',
        type: 'Multiple',
    };
}
public rowSelected(args: any) {
    let selectedrowindex: number[] =
this.ganttObj.selectionModule.getSelectedRowIndex(); // get the selected
row indexes.
    alert(selectedrowindex); // to alert the selected row indexes.
    let selectedrecords: Object[] =
this.ganttObj.selectionModule.getSelectedRecords(); // get the selected
records.
    console.log(selectedrecords); // to print the selected records in
console window.
}
}

```

### MAIN.TS

```

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

```

### Multiple Selection based on condition

You can select multiple rows based on condition by using the [selectRows](#) method.

In the following code, the rows which contains **TaskId** value as 3 and 4 are selected at initial rendering.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';

```

```

@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [selectionSettings]="selectionSettings"
    (dataBound) = "dataBound($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public selectionSettings?: SelectionSettingsModel;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      }
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',

```



```

        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.selectionSettings = {
        mode: 'Row',
        type: 'Multiple',
    };
}
public dataBound(args: any) {
    var rowIndexes: any[] = [];
    this.ganttObj.treeGrid.grid.dataSource.forEach((data: any, index:
any) => {
        if (data.TaskID === 3 || data.TaskID === 4) {
            rowIndexes.push(index);
        }
    });
    this.ganttObj.selectRows(rowIndexes);
}
}

```

### MAIN.TS

```

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

```

### Touch interaction

When you **tap** gantt row, tapped row will be selected.

**Single selection** : To select a single row or cell, perform **single tap** on it.

**Multiple selection** : To perform multiple selection, **tap** on the multiple selection popup, and then tap the desired rows or cells.

![[Multiple selection]](images/multiple-selection.PNG)

See Also

- [Touch interaction](#)

### Rows in Angular Gantt component

Row represents a task information from the data source, and it is possible to perform the following actions in Gantt rows.

#### Row height

It is possible to change the height of the row in Gantt by setting row height in pixels to the [rowHeight](#) property. The following code example explains how to change the row height in the Gantt at load time.

### APP.COMPONENT.TS

```

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

```

```

import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [rowHeight]="60"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    }
  }
}

```

```

    };
  }
}

```

## MAIN.TS

```

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

```

### Expand/Collapse Row

In Gantt parent tasks are expanded/collapsed by using expand/collapse icons, expand all/collapse all toolbar items and by using public methods. By default all tasks in Gantt was rendered in expanded state but you can change this status in Gantt.

#### *Collapse all tasks at Gantt load*

All tasks available in Gantt was rendered in collapsed state by setting [collapseAllParentTasks](#) property as **true**. The following code example shows how to use [collapseAllParentTasks](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [collapseAllParentTasks]='true'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
};
}
}

```

## MAIN.TS

```

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

```

### Define expand/collapse status of tasks

In Gantt, you can render some tasks in collapsed state and some tasks in expanded state, this can be done by defining expand status of the task in data source. This value was mapped to Gantt component by using [expandState](#) property. The following code example shows how to use this property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
    imports: [
        GanttModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template:
      `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskSettings"></ejs-gantt>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    // Data for Gantt
    public data?: object[];
    public taskSettings?: object;
    public ngOnInit(): void {
      this.data = [
        {
          TaskID: 1,
          TaskName: 'Project Initiation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isExpand:true,
          subtasks: [
            { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
            { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          ]
        },
        {
          TaskID: 5,
          TaskName: 'Project Estimation',
          StartDate: new Date('04/02/2019'),
          EndDate: new Date('04/21/2019'),
          isExpand:false,
          subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
          ]
        },
      ];
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        expandState:'isExpand',
        child: 'subtasks'
      };
    }
  }
}

```

**MAIN.TS**

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

### Customize expand/collapse action

On expand action [expanding](#) and [expanded](#) event will be triggered with current expanding row's information. Similarly on collapse action [collapsing](#) and [collapsed](#) event will be triggered. Using this events and it's arguments you can customize the expand/collapse action. The following code example shows how to prevent the particular row from expand/collapse action using [expanding](#) and [collapsing](#) event.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" (collapsing)="collapsing($event)"
    (expanding)="expanding($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
```

```

        endDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
              startDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', startDate: new Date('04/04/2019'),
              Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', startDate: new Date('04/04/2019'),
              Duration: 3, Progress: 50 }
        ]
    },
];
this.taskSettings = {
    id: 'TaskID',
    name: 'TaskName',
    startDate: 'StartDate',
    endDate: 'EndDate',
    duration: 'Duration',
    progress: 'Progress',
    child: 'subtasks'
};
}
public collapsing(args: any){
    if(args.data.TaskID==1)
        args.cancel=true;
};
public expanding(args: any){
    if(args.data.TaskID==5)
        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 rows and cells

While rendering the TreeGrid part in Gantt, the [rowDataBound](#) and [queryCellInfo](#) events trigger for every row and cell. Using these events, you can customize the rows and cells. The following code example shows how to customize the cell and row elements using these events.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
    imports: [
        GanttModule
    ],

```

```

standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
[taskFields]="taskFields" [columns] = "columns" [splitterSettings] =
"splitterSettings" (rowDataBound) = "rowDataBound($event)" (queryCellInfo)
= "queryCellInfo($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskFields?: object;
  public columns?: object[];
  public splitterSettings?: object;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskFields = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.columns = [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '150' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width: '150' },
    ];
    this.splitterSettings = {
      columnIndex: 3
    };
  }
  public queryCellInfo(args: any) {
    if (args.column.field == "Progress") {
      if (args.data.Progress < 25)
        args.cell.style.backgroundColor="lightgreen"
      else
        args.cell.style.backgroundColor="yellow"
    }
  };
  public rowDataBound(args: any) {
    if(args.data.TaskID==4)
      args.row.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 rows

You can customize the appearance of a row in grid side, by using the [rowDataBound](#) event and in chart side by using [queryTaskbarInfo](#) event

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskFields" (queryTaskbarInfo) = "queryTaskbarInfo($event)"
    (rowDataBound) = "rowDataBound($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskFields?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location', StartDate:
            new Date('04/02/2019'), Duration: 0, Progress: 50,isParent:false },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate: new
            Date('04/02/2019'), Duration: 4, Progress: 70, resources: [2, 3,
            5],isParent:false },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate: new
            Date('04/02/2019'), Duration: 4,Predecessor:"2FS", Progress:
            80,isParent:false },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        isParent:true,
```

```

        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for estimation',
StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50, resources:
[4],isParent:false },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 70, resources: [4,
8],isParent:false },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate: new
Date('04/04/2019'), Duration: 0,Predecessor:"6SS", Progress: 50, resources:
[12, 5],isParent:false }
        ]
    },
];

    this.taskFields = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
}

public queryTaskbarInfo(args: any) {
    if (args.data['TaskID'] == 4) {
        args.rowElement.style.background = 'cyan';
    }
}

public rowDataBound(args: any) {
    if (args.data['TaskID'] == 4) {
        args.row.style.background = 'cyan';
    }
}
}
}

```

### MAIN.TS

```

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

```

### Styling alternate rows

You can change the background colour of alternative rows in Gantt chart, by overriding the class as shown below.

```

`css

.e-altrow, tr.e-chart-row:nth-child(even) {

background-color: #f2f2f2;

}
`

```

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 6, TaskName: 'Develop floor plan for
            estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 7, TaskName: 'List materials', StartDate: new
            Date('04/04/2019'), Duration: 3, Progress: 50 },
          { TaskID: 8, TaskName: 'Estimation approval', StartDate:
            new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
      },
    ];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',

```

```

        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
}
}

```

## MAIN.TS

```

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

```

## Row spanning

Gantt chart has an option to span row cells. You can achieve this using [rowSpan](#) attribute to span cells in the [QueryCellInfo](#) event.

In the following demo, **Soil test approval** cell is spanned to two rows in the **TaskName** column.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GanttModule } from '@syncfusion/ej2-angular-gantt';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" (queryCellInfo)="queryCellInfo($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
            StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
            new Date('04/02/2019'), Duration: 4, Progress: 50 },

```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
    ],
    };
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks'
    };
}
public queryCellInfo(args: any) {
    if (args.data['TaskID'] == 4 && args.column.field === 'TaskName') {
        args.rowSpan = 2;
    }
}
}
}

```

## 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 and it can be defined by the [columns.clipMode](#) property.

The following are three types of `clipMode`:

- **Clip**: Truncates the cell content when it overflows its area.
- **Ellipsis**: Displays ellipsis when content of the cell overflows its area.
- **EllipsisWithTooltip**: Displays ellipsis when content of the cell overflows its area; it displays the tooltip content when hover over ellipsis.

## NOTE

By default, all the column's [clipMode](#) property is defined as `EllipsisWithTooltip`.

*Cell tooltip*

You can enable or disable the Grid cell tooltip using the [columns.clipMode](#) property.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.columns = [
      { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
      { field: 'TaskName', headerText: 'Task Name', width: '150',
clipMode: 'EllipsisWithTooltip' },
      { field: 'StartDate', headerText: 'Start Date', width: '150' },
      { field: 'Duration', headerText: 'Duration', width:
'150',clipMode: 'Clip' },
      { field: 'Progress', headerText: 'Progress', width: '150' },
    ];
  }
}
```

**MAIN.TS**

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

## Export

### Excel export in Angular Gantt component

Gantt supports client-side exporting, which allows you to export its data to the Excel and CSV formats. Use the [excelExport](#) and [csvExport](#) methods for exporting. To enable Excel export in the Gantt, set the [allowExcelExport](#) to true.

To export data to Excel and CSV, inject the `ExcelExportService` in the provider section of `AppModule`.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, ExcelExportService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent, ToolbarItem } from '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { GanttData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, ExcelExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
    (toolbarClick)="toolbarClick($event)" allowExcelExport='true'></ejs-
    gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent;
  public ngOnInit(): void {
    this.data = GanttData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
```

```

        child: 'subtasks'
      };
      this.toolbar = ['ExcelExport', 'CsvExport'];
    }
    public toolbarClick(args: ClickEventArgs): void {
      if (args.item.id === 'ganttDefault_excelexport') {
        this.ganttObj!.excelExport();
      } else if (args.item.id === 'ganttDefault_csvexport') {
        this.ganttObj!.csvExport();
      }
    }
  };
}

```

## MAIN.TS

```

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

```

## Export

### *Pdf export in Angular Gantt component*

PDF export allows exporting Gantt data to PDF document. You need to use the [pdfExport](#) method for exporting. To enable PDF export in the Gantt, set the [allowPdfExport](#) to true.

To export data to PDF document, inject the PdfExport module in Gantt.

Note: Currently, we do not have support for exporting manually scheduled tasks.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, PdfExportService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { GanttComponent, ToolbarItem, SelectionSettingsModel } from
 '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
    (toolbarClick)="toolbarClick($event)" allowPdfExport='true'></ejs-
gantt>`,

```



```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent{
        // Data for Gantt
        public data?: object[];
        public taskSettings?: object;
        public toolbar?: ToolbarItem[];
        @ViewChild('gantt', {static: true})
        public ganttChart?: GanttComponent;
        public ngOnInit(): void {
            this.data = editingData;
            this.taskSettings = {
                id: 'TaskID',
                name: 'TaskName',
                startDate: 'StartDate',
                duration: 'Duration',
                progress: 'Progress',
                child: 'subtasks'
            };
            this.toolbar = ['PdfExport'];
        }
        public toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'ganttDefault_pdfexport') {
                this.ganttChart!.pdfExport();
            }
        }
    };
}

```

## MAIN.TS

```

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

```

### Indicators in PDF exporting

The PDF export functionality allows users to export Gantt charts enriched with dynamic indicators and accompanying images.

These indicators, represented by images, can be effortlessly defined using the [base64](#) encoding value in the data object of datasource. This data object field should be mapped to indicator property of [task fields](#).

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, PdfExportService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { GanttComponent, ToolbarItem, SelectionSettingsModel } from
 '@syncfusion/ej2-angular-gantt';

```

```

import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'></ejs-
gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  @ViewChild('gantt', {static: true})
  public ganttChart?: GanttComponent;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.toolbar = ['PdfExport'];
  }
  public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttDefault_pdfexport') {
      this.ganttChart!.pdfExport();
    }
  }
};
}

```

## MAIN.TS

```

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

```

### Exporting Gantt data as a blob object

In Gantt, you can export the Gantt chart data as a blob object, which allows you to preview or modify the data before exporting it.

To export the Gantt chart data as a blob object, follow these steps:

step 1: pdfExport fourth argument set as `true`.

step 2: Then , `pdfExpComplete` return as blob object.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule, ExcelExportService, ExcelExport, GanttAllModule } from
 '@syncfusion/ej2-angular-gantt'
import { ToolbarService, PdfExportService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt, Toolbar, PdfExport, Selection, ExcelExportService,
 PdfExportService,
   ToolbarService, ToolbarItem, GanttComponent } from '@syncfusion/ej2-
 angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-
 navigations/src/toolbar/toolbar';
import { ExcelExportCompleteArgs, PdfExportCompleteArgs,
 SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule, GanttAllModule
  ],
  providers: [ToolbarService, PdfExportService, SelectionService,
 ExcelExportService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
 [dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
 (toolbarClick)="toolbarClick($event)"
 (excelExportComplete)="excelExpComplete($event)"
 (pdfExportComplete)="pdfExpComplete($event)" allowPdfExport='true'
 allowExcelExport='true' [treeColumnIndex]="1"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None,
  providers: [ToolbarService, ExcelExportService, PdfExportService]
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  @ViewChild('gantt', {static: true})
  public ganttChart?: GanttComponent;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    }
  }
}
```

```

    };
    this.toolbar = ['PdfExport', 'ExcelExport'];
  }
  public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttDefault_pdfexport') {
      this.ganttChart!.pdfExport(undefined, undefined, undefined, true);
    }
    if (args.item.id === 'ganttDefault_excelexport') {
      this.ganttChart!.excelExport(undefined, undefined, undefined, true);
    }
  };
  excelExpComplete(args: ExcelExportCompleteArgs) {
    // This event will be triggered when excel exporting.
    args.promise!.then((e: { blobData: Blob }) => {
      // In this `then` function, we can get blob data through the
      arguments after promise resolved.
      this.exportBlob(e.blobData);
    });
  }
  pdfExpComplete(args: PdfExportCompleteArgs) {
    // This event will be triggered when pdf exporting.
    args.promise!.then((e: { blobData: Blob }) => {
      // In this `then` function, we can get blob data through the
      arguments after promise resolved.
      this.exportBlob(e.blobData);
    });
  }
  public exportBlob = (blob: Blob) => {
    const a: HTMLAnchorElement = document.createElement('a');
    document.body.appendChild(a);
    a.style.display = 'none';
    const url: string = (window.URL as any).createObjectURL(blob);
    a.href = url;
    a.download = 'Export';
    a.click();
    (window.URL as any).revokeObjectURL(url);
    document.body.removeChild(a);
  }
}

```

### MAIN.TS

```

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

```

#### Single page exporting in gantt

In Gantt, we have provided support to export the Gantt component where each rows are auto-fit to the PDF document page width by setting [isFitToWidth](#) as true in `fitToWidthSettings` of `PdfExportProperties`.

Also, we can customize the chart width and grid width in exported file using [chartWidth](#) and [gridWidth](#) by defining it as percentage in string.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService, PdfExportService, SelectionService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { GanttComponent, ToolbarItem, PdfExportProperties } from
 '@syncfusion/ej2-angular-gantt';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'></ejs-
gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  @ViewChild('gantt', {static: true})
  public ganttChart?: GanttComponent;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks',
    };
    this.toolbar = ['PdfExport'];
  }
  public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttDefault_pdfexport') {
      let exportProperties: PdfExportProperties = {
        fitToWidthSettings: {
          isFitToWidth: true,

```

```

    }
    };
    this.ganttChart!.pdfExport (exportProperties);
  }
};
}

```

## MAIN.TS

```

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

```

### Exporting with templates

#### Exporting with column template

The PDF export functionality allows to export Grid columns that include images, hyperlinks, and custom text to an PDF document using [pdfQueryCellInfo](#) event.

In the following sample, the hyperlinks and images are exported to PDF using [hyperlink](#) and [image](#) properties in the [pdfQueryCellInfo](#) event.

Note: PDF Export supports base64 string to export the images.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
  ToolbarService,
  PdfExportService,
  SelectionService,
} from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import {
  GanttComponent,
  ToolbarItem,
  PdfExportProperties,
} from '@syncfusion/ej2-angular-gantt';
import { editingResources } from './data';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
@Component({
  imports: [ GanttModule],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-gantt #ganttDefault id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
(pdfQueryCellInfo)="pdfQueryCellInfo($event)"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'
[allowResizing] = 'true' rowHeight='50'

```

```

[splitterSettings]="splitterSettings" [resourceFields]="resourceFields"
[resources]="resources">
  <e-columns>
    <e-column field='TaskID' headerText='Task ID' textAlign= 'Left'
width= 100></e-column>
    <e-column field='TaskName' headerText= 'Task Name' width=
150></e-column>
    <e-column field= 'resources' headerText= 'Resources' width=
250>
      <ng-template #template let-data>
        <div *ngIf="data.ganttProperties.resourceNames">
          {{data.ganttProperties.resourceNames}}
        </div>
      </ng-template>
    </e-column>
    <e-column field= 'EmailId' headerText='Email ID' width= 150
></e-column>
  </e-columns></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public resources?: object[];
  public toolbar?: ToolbarItem[];
  @ViewChild('ganttdefault', { static: true })
  public ganttChart?: GanttComponent;
  public resourceFields?: object;
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Product concept',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        EmailId: 'MartinTamer@gmail.com',
        subtasks: [
          {
            TaskID: 2,
            TaskName: 'Defining the product and its usage',
            StartDate: new Date('04/02/2019'),
            Duration: 3,
            Progress: 30,
            resources: [2],
            EmailId: 'RoseFuller@gmail.com',
            resourcesImage:
'/9j/4AAQSkZJRgABAQAAQABAAD//gAfQ29tcHJlc3NlZCBieSBqcGVnLXJlY29tcHJlc3P/2wC
EAAQEBAQEBAQEBAQGBgUGBgghBwcHCAwJCQkJCQwTDA4MDA4MExEUEA8QFBEeFxUVFx4iHRsdIio
lJS00MjRERFwBBAQEBAQEBAQEBAQYGBQYGCACHBwcIDAKJCQkJDBMMDgwMDgwTERQQDxAUER4XFRU
XHiIdGx0iKiUlKjQyNEREXP/CABEiADcAnwMBIgaCEQEDEQH/xAAbAAADAAMBAQAAAAAAAAAAAAA
FBwgEBgkCA//aAAGBAQAAAC/hQFOvYjnCinKzbmZbGH5zuQtL+rjE/f05y7I93/rpMhES5qCgxO
TPERMqDaDCzVpNoBsPfbf/8QAGgEAAQUBAIAAAAAAAAAAAAAAAAAAAECawQFBv/aAAGBAhAAAAAOWZj
mNLVM6a2Pan//xAXAQEBAQEAAAAAAAAAAAAAAAAABAUG/9oACAEDAAAAGjNO7PFxm1FEH//xAA
3EAAcAgECBAMFBgQHAAAAAAAAABAgMEBQAGBxESQSExMhATUVKBCBQiYWKhFiNzkTNCU2RygrH/2gA

```

```
IAQEAAT8A0chavSvWwcaFUYrJdlBMSkeYjA9Z/bW5b209pY98xvncBff57UrKrP8ACOGL1H8gCdR
cfOB8txaopTojeU5p8o9Uq+OuVUv7XzrLE4DIYpvnNduY+Vif2I1VyK0NiPH5eBYLD+EUqEmCc/B
SfJv0n2ZB5MjajwLZ2RCnvbkinkViJ5CMH5n/APNdNajV5L0Q14IyflREUeJlVDP53jTu65l72QM
OMWZ4MbW/yQwBuw+Yj1HW3OAEF1lntZ50iHNRGkHiSe/MtrbEF3ghuPEWkyktvbt2daeQRx4oH8E
fkPk1PTr5CrLVtRBom5fkQR3B7EdjrD2Z1exibrdVury/mf6sLeiT+vY6wRV69rJv671mSX4n3an
oRfoo1l6pv4rKUAWQ2ak8AY+QMiFef76x2VbacmNrvjnnmjAMiGRU5OW9IB8W0tucRXk2ra3FiMK
9panISQGTpCv+ZAJ1ZvZjiJgbr28VBVimjmmj6RYVo2V/DwljQN3BI1Gysqub5hgCNZ2VcbZx2Z7
Rl683LzaORSw/syJW3HUYHffEVkB9m7sNitqby3LVzlFmkhlkmrFVKO6MSY+nXBvN0Zq+YoLQsix
LKr9DxosBHkSCxAlXvrFTRzyDCrSjhnM6x9KgCFwT6l5dtIOLFX4Aa3uhG3bCjxYyxfU9WsEfu5v
4lvBqlhygPeGY9aH9yPZ9rHEQ0M5tvOo/4sjVnqSoP9uQQw+kuuFMAd0DW4pK5J61lkYsOf8A28D
rYaU23dFVqoohgWWdlTyDEcv7nnz9mShTIZGhiTzaNfelZ5dlAKIPqTrK1bEU8GYx8ZezApSWIec
8BPMqP1DzXXEn7Ue2dlT2sNisLfyGzi7TxmrWT+rP+JtY7c03GVty/wAVSKcnNcjsQyJ4CCLoEaJ
EOypyltjgruGnuypiZcpXkSMMWVevZVHeH5mTnzGt75ylwWweJkw5jnzlu5FyD94IiGm+jenWy+N
G1N60m+4CxHl04laTGshMhZjyHQw8GBOsZTmrJNaukNftMJJyPJeyxj9KD2cReDu00JNUnJVBDdH
+Hai/C6nW2+AWe4ZbrOTe3VvYKeKSByT0ypzIKkL3l1tfZ+8It62tx5a37h6+T+/0pY5FKycj0CAg
EkRmPwlujg/n+Ke7XzuRvpTwcaJBVjRg0vuk8T/xJ0tmcPtU7EpJVw9VRL0/zJ28XY+z/xAAiEQa
CAQMEAgMAAAAAAAAAAABAgMABBEQEHNRISIFQYH/2gAIAQIBAT8AqW/hjk4y/tlioJ0nTemtyA0
pYREOjeT3XxjFufxhcg/ut5aMw5Ez7H6Gas7ZraHa4wzHcdVYjaOjmiSTk6f/xAAiEQACAQIGAwE
AAAAAAAAAABAgMEEQAFEBITISJBZUH/2gAIAQMBAT8AxFltRJGsvH4H3fFTTTPsf2LjWlBECh
plaNk8V+YzdVUwDddrH81y6t4xxkjpSLsbdYr6hamfehuoULr91//2Q==',
},
{
  TaskID: 3,
  TaskName: 'Defining target audience',
  StartDate: new Date('04/02/2019'),
  Duration: 3,
  resources: [3],
  EmailId: 'MargaretBuchanan@gmail.com',
  resourcesImage:

  '/9j/4AAQSkZJRgABAQAAQABAAD//gAfQ29tcHJlc3NlZCBieSBqcGVnLXJlY29tcHJlc3P/2wC
EAAQEBAQEBAQEBAQGBgUGBgggHBwcHCAwJCQkJCQwTDA4MDA4MExEUEA8QFBEeFxFxUVF4iHRSdIio
lJS0MjRERFwBBAQEBAQEBAQEBAYGBQYGCACHBwcIDAKJCQkJDBMMDgwMDgwTERQQDxAUER4XFRU
XHiIdGx0iKiUlKjQyNEREXP/CABEiADcAnwMBiGACEQEDEQH/xAAZAAACAwEAAAAAAAAAAAAAAAAA
EBwMFCAB/2gAIAQEAAAAAAAA39UUCnct2dVSZMV1l1tS5G5fmOGS73fU8SeVXNvlpWZ6WVEOtoiTVwgB
tW2poSoCztszrv//EABoBAAICAwAAAAAAAAAAAAAAAAAUGAAMBAgT/2gAIAQIAAAAA1WTBChM72vC
cdKyi+f/EABkBAAlDAQAAAAAAAAAAAAAAAAAGAQMFBP/aAAGBAxAAAACWjF4L3/LTR/WMcvoP/8Q
AHhAAAgMBAQEBAQEAAAAAAAAABAUCAwYBAACIFBP/2gAIAQEAAQgA9NkSZZMdLpDUWdEkXoQvreH
KL6PcoksbBcYZYVpdSRWvb+YTsYlVpBnDIPOJCzOPgtNuibSb2HzhkuvnFDL6jW4QvhA+cdL9xmR
y7k5N8ZkKTUnOWDkMPfTuXV5uu6Fd4kuli8006greD+1lkIrb/fmd1ezrr//AFe2xWkrnPs5OPE
Krz2iN6oyEy0QFzWhh0hfI921rv0iT+1UQt/Pau5PlTed2808zpEeIu/z9PUyvqjdTbXNgvOGvt
7WSFfW9p7SKttlBxagJEsilsKE0GVMGDEBT1qKRVfQ4XhmDnj1lC6r+jhxXQ24moaF3DA06BJ8S7
RodKp2KBzn12lTLA7hoXFG+KTy5fM5V1No1T3QMdC4eQHVkQQL/zZu/oDmS37/B/OM189WURkfV/
EADMQAAlBAwIDBQYFBQAAAAAAAAEACAwAEERiHmUGBBRMyUWEQFCIzYpEGi0JxsVNjc4Kh/9oACAE
BAAk/AKRCqMvkVZQTEpHERgeMj7Ve3faM5BKQazlz9EaFVA9TX4TW2USmLL41huWQBsTywa7ZmhX
O8eszRBseFo5OHsOfguH2ilQkwTnyUng30n2OyIU728kU4KxE4EYPJn/ioglrZwEhEHEjZUGPM0G
jzrC3gk/fka8V7cSS+vdqdEa9AKULE7RsTOv9ppgtSvFK65QGJwrfSXGKkd5TvoSJ3PXSDispokX
DEEFaZ3WC7iVC3m0e9ciY9vNji0cilh9mUVVxFsgPSlJQgA+u+aiBvLfLkKwythv44UzM07ZUM5DL
obPwHl64qUhQqgM7FiApzueJxVsYYJLoiJTzKFFZq3YyxdTqrZrS4coDzhmOtD0yRXB1KnrUg7j4
xOipmbXsAyljjAHLfMY7cOXuJp4wjkafoCnz5mptFxeq0MMh/Qxyxb/UCpmlWys4Ldpn8UpiQKXb
1bGay0aK9lC45KAUQdSc9KjL3MClJYhxngJyVH1DitSao26EEcQRyIqFJm0g6C2nlyNWJU4t4zqyq
g+bVete9qTZSGxtYnBixw2nP3LGrxb22v1U2yQ7yO7D5Wnk44MDwohr+6YST44LgYWMfSg9lX7rd
v8zI1Qzf5E8/UblMs3Z1/cLPbe7sCYSI1jMY1EHTtkVaLNd6CIIflqXI21M3rxNdvQRRzPulq+sq
nKNCfBVoBoyTK27Fm4nfmfZ//xAAjEQACAQQAUAUAAAAAAAAAABAgMABBESEAUiURmjMWKR/9o
ACAECAQE/AGZUVnY4UDJNTdbLSN6b6R5wO3Oasr+O6ULkb44uRtbzD6GpY4cIjKdfIrouqzKoBz3
fnBAIIPwauk0Y2scezByB5rpVmbWD3FxiEY7eGJ3dEGZEknn/xAAjEQACAgEEAQUBAAAAAAAAAA
```



```

BAGMEEQAFEBIiITFRYnGx/9oACAEDAQE/AFUsyqoySQANQbATEC695CMkdsY1eoS03IYeGeKx62IT911VE3m6MpPwdb/E5rvMWUAdR+kngHBBGqFvNYSysFygJPtrebot2AInzEv95ksZsqiO56qAAB6Dn//Z',
    },
    {
        TaskID: 4,
        TaskName: 'Prepare product sketch and notes',
        StartDate: new Date('04/02/2019'),
        Duration: 2,
        Predecessor: '2',
        Progress: 30,
        resources: [4],
        EmailId: 'FullerKing@gmail.com',
        resourcesImage:

        '/9j/4AAQSkZJRgABAQAAQABAAD//gAfQ29tcHJlc3NlZCBieSBqcGVnLXJlY29tcHJlc3P/2wCEAAQEBAQEBAQEBAQGBgUGBggbHBwcHCawJCQkJCQwTDA4MDA4MExEUEA8QFBEeFxFVFX4iHRsdIio1JS00MjRERFwBBAQEBAQEBAQEBAQYGBQYGCACHBwcIDakJCQkJDBMMDgwMDgwTERQDxAUER4XFRUXHiIdGx0iKiUlKjQyNEREXP/CABEiADcAnWMBiGACEQEDEQH/xAAbAAACAwEBAQAAAAAAAAAAAAAABwQGCAMCCf/aAAGBAQAAADfwhV0x/EZ4hW5npVo+hcTlnMn4TW6ofZUBIXDSIEEnOzwAaDYEyICYV79vc+aEqNLsbBM//8QAGAEAAwEBAAAAAAAAAAAAAAAAAAGQFAwD/2gAIAQIAAAABNvRaHSpjAqO9hof/8QAGQEAAgMBAAAAAAAAAAAAAAAAAAGUDBAYA/9oACAEDeAAAADbLIbutRIi2OdXdagD/xAAfEAACAwEBAQADAQAAAAAAAAAEBQIDBgEHABITFBX/2gAIAQEAAQgA+/0izrLKEUj9081zxtwL2r3Pze87lUFfwDEEVr1Q21sL617X47th5VaUf2TSLZ/00Z9IT468vpqspb+dtFYNHQ3jnpLFFP61zAoQY4aY5acm+MyFJqOf5jlMu+12WWaxaJBjugEFv8cG+5kMlobU70irSIgntauc+rF0rHtsVpK5z9nZ/ihVc56fmiTmueaDmJmVJkTeVxnnCKIgcJUrma0oRaHRwYYYeO3h3mdIjxF3+fp6mRlA5dMxyHyYqbZuDVFcBtyFFuAVdpIlKz5lTBgxAU9aikV30OALTIDMbnQ9akbk12vEoL7YmEli9xrWtZKdU6tYkvF7IVYHCNC4o35zmhjLJmCfqbZwqBJeqGzphcDs2mzjIsTN8WJ84Ak7K2H3/8QAMxAAAQEDAgMECQMFAAAAAAAAAAQIDAAQREjETIVEFQWGHQBjMkJicXKBBiIzULOCKbH/2gAIAQEACT8ApE0IxWS9l5xKRuEHxnyrt687Wv4v5o4Q7oh8Fj0pX6PuY7HIBunjjDjqdCk12xItvOgkhZJDPbsph9D58iDUKw3D8opUJME6KtS3yn0SMql0LeyqcFYicCNT1f/AJXsZryMxkxjBSHbC/dUS3XEjDSaSoYN/kRVndZikKSppJKMOoHcakkXseadI7mCbKhdZwZUB2K0gaKTGANwe4g9xFNqu7XHtP7sLe5J9e40My31zJL1IjU6EH4UUTxNFsR0KvIVxVnLPPGig5dIkzjnguQTIuxxi4bHtpAgBHZYNWcEc9rexJrt5lnDRzciCQBUgaQ20RYg55ledbRl7ebG7RyKWH+mUVv6sgPhgUoYiW03lDclVUBiqSfrUEMlyrH4QWOFOpFdopjmAxMMN0C489qgRtckMiRON3WQPjyonTFEQDPRRiubGWL8nVXJrS4coOsMx1ofMiow6HcVcvBLLMdMyAF10/xCnLXnMLcNAMdNVScd7ePEkxULqZ+/C4A9GWjRXurnHcoBRB+Saj13MAKSxDea8yo+YbrUmt2HMEEAfr4ipHkljYm4Qe+veGHUVLFpZsamWpGnyWSd1HvTIobC/aDtUrTTNgJbqhEruTyUCiGv7phJPjZcDCxj5UHok9WvH95gMxyfevXxFW4MWzNFIGR1+hwaaw2u5c8UJEBfkjGcMVw3iKs/Ubaxt7mOX1iZzuJLOysZMpgs37eg3ocW9fOudhjGdwg7h6P/8QAIREBAAIIBBAEFAAAAAAAAAAAAAAAAAQIDAAQQETESBSJBYYH/2gAIAQIBAT8AnMhFk4a2Euph+ZXYWG2rOaX6yTxEDr5z015jMD2m0kB5yyLXKzkAVTND4RoiHart3l1NdkGE48mRhGIeMTonv//EACIRAQACAQMEAwEAAAAAAAAAAAECAxEAEIEBTfHfSEiNP/aAAGBAwEBPwCuuVs4wj5dfFTBzFUPGdXUyplxdu3/ANURQyQq6xCUs5cYdd7hwnUyRk7R5cjj510t50qGP2kQfSa7k2T6qyUj6MB62FETVHU3U2RsrniXh96nZOaspLlV2//Z',
    },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Concept approval',
        StartDate: new Date('04/02/2019'),
        Duration: 0,
        Predecessor: '3,4',
        resources: [5],
        EmailId: 'Davoliofuller@gmail.com',
        resourcesImage:
    }

```

```
'/9j/4AAQSkZJRgABAQAAQABAAD//gAfQ29tcHJlc3NlZCBieSBqcGVnLXJlY29tcHJlc3P/2wC
EAAQEBAQEBAQEBAQGBgUGBggHBwcHCawJCQkJCQwTDA4MDA4MExEUEA8QFBEEfxUVF4iHRsdIio
lJS00MjRERFwBBAQEBAQEBAQEBAQYGBQYGCACHBwcIDAKJCQkJDBMMDgwMDgwTERQQDxAUER4XFRU
XHiIdGx0iKiUlKjQyNEREXP/CABEiADcANwMBiGACEQEDEQH/xAAaAAACAwEBAAAAAAAAAAAAAAAA
HCAQFBgID/9oACAEBAAAAAH+qsdiSrczqiUBMb22cXitDeKqHJh+9YFRxtGA870evw2Kx1qZOdFq
/gVwmsmjsTDfKu9sf/8QAGQEAAwEBAQAAAAAAAAAAAAAAAAAQFBgED/9oACAEECAAAAOraDfjmQNo
zlhsZWP/EABkBAAIDAQAAAAAAAAAAAAAAAAAGAwQFAf/aAAGBAXAAADu/10535dXB6VKBDMf/8Q
ANRAAAgIBAgQDBgUCBwAAAAAAAAQIDBAUABhESIUEHIjEQEzJhgZEIFFJicRYjQkNRU6Gx0f/aAAG
BAQABPwDRyFq9K9bBxoVRisl2UEXKR6iMD4z/AMA3ZuPa00ljG6stav3JFLpVVizn5iKMqqL821S
8WPC675v6duQR8SDK9VOA+fkYnWJfE5eimT2jnZFhPoFcywhv0vHJ1U/Y6qZSeGePHZeAQWHPCKV
OsE5/0Un0b9p9mRle9ZTDV5GSMoJLkingViJ4CNT+p/8ArWSsLicRZkpQKErQkRRjopb0UfxqvtK
DJy5DI5ZFnuW3LyyP1PA/4RrM7OxVPnaJCir5Qo462Bck2luuCerZc46+Vgnic9FPz/j1GrVWtfr
SVbKB42HXsQexB7Eaw9mdXsYm63Nbq8P7n+7C3wSfz2OsEVkr2snJle9ZklHc+7Q8kY+gGt3OYMR
zMQqvNGp+p1l/ELD7fdaj15ZmLBCyPGoB+QZgW+glu/etWq8cEVI2PeRrM3K4H1ZebsCTrbmWpZq
JrEdaeKe0VeEfu3kC9QOYlAeH1li7iXMXRlSVZCY1DFTxBZeh+xlnZVxtnHZntGXrzcPVo5FLD7M
o1txlGCxRHxCsgOt2442sM3lDGJxMVb0IH/gPHVuth1sloI4meRuMrhVBVFXuen2lubKY07gi9yv
vVSFIDGkbIyIo9OLDh/AltCCNoMpdqM0R/J2EjUgcyMqqg3A62nhzg8BjMbJIXkih4yv+qWQl3I+
XMTw1vdCNu2FHVjLF9TzawZ/Lm/iWPBqdhypGeGY86EfcjUsayK6OoZWUqQfQg6yuCWPMGvYkmik
qWRMhjdk94i9VDcpHMPHqNbrxUty4VmhgEDyBnMMUkbso7czSPw902vDivEk/AgK5aOMD08yedvt
zBfZkoUyGRoYk8WjRXtWeHZQCid6k6ytWxFPBmMfGXswKuliHrPATxKj9w9V1mfETY23KkV3Ob1p
1VdOdYGYtYI90kKcX1Z3zQ8Q6lj07TpzitjbULMTyoEaduRX4qvZRx6cdbq3Nu5bDKFPkPUiuUCB
evEknhrJ+J+6bNqEUMrNDFWsCdJUYq7zIxYScfkTxGvCn8S2e3ZNits39ny5LKjyWr9SURx8naVo
yvRtYynNWSaldIa/aYSTkei9ljH7UHs8QPCTa+/4C1+uIbw+CzF5XGsNsbdXhdtN14PEClfju5SS
ylqySAsRijjCKilSX8vxa8RU35ubBxbdwWBBrQGy5F65+ZQNLGT0iHE8VX9etl/hN3LlpIbe5MrWq
0+6Vn530theGelvdvHpSwNBek/wAyc9Xdu5JPs//EACMRAAIBAwMEAwAAAAAAAAAAAAAECawAREgQ
QMQUtIUEim3L/2gAIAQIBAT8AJABJ4FSdWChmwOPANaPVJq4s15HO0/0y/klIUkHcSUHqulKEzwS
ysLnzvpBJFM0CDk3Xx6NaOJooQrizHfJTAD2o8wbCS3ytv//EACMRAAEDAwQCAwAAAAAAAAAAAAA
CAxEABBI FECEXFCJBUXH/2gAIAQMBAT8AAJIA7JimNELsAujOORV7Zrs3i0v9B2tY8liTHu06t7V
cyk4qUJyNa8BnC3cnEKCRA7ESd9OvG3rYXD6uQnE8/IrUXkv3K1oVkn73BcDpSHlhCuSifUnf/9k
=',
},
{
  TaskID: 6,
  TaskName: 'Market research',
  StartDate: new Date('04/02/2019'),
  EndDate: new Date('04/21/2019'),
  EmailId: 'Vanjack@gmail.com',
  subtasks: [
    {
      TaskID: 7,
      TaskName: 'Demand analysis',
      StartDate: new Date('04/04/2019'),
      EndDate: new Date('04/21/2019'),
      EmailId: 'RoseFuller@gmail.com',
      subtasks: [
        {
          TaskID: 8,
          TaskName: 'Customer strength',
          StartDate: new Date('04/04/2019'),
          Duration: 4,
          Predecessor: '5',
          Progress: 30,
          resources: [7],
          EmailId: 'Fullerbuchanan@gmail.com',
          resourcesImage:

```

```
'/9j/4AAQSkZJRgABAQAAQABAAD//gAfQ29tcHJlc3NlZCBieSBqcGVnLXJlY29tcHJlc3P/2wC
EAAQEBAQEBAQEBAQGBgUGBggHBwcHCAwJCQkJCQwTDA4MDA4MExEUEA8QFBEeFxUVFx4iHRsdIio
lJS00MjRERFwBBAQEBAQEBAQEBAQYGBQYGCACHBwcIDAKJCQkJDBMMDgwMDgwTERQQDxAUER4XFRU
XHiIdGx0iKiUlKjQyNEREXP/CABEiADcANwMBiGACEQEDEQH/xAAaAAACAgIDAQAAAAAAAAAAAAAA
EBwUIAwYAAQKc/9oACAEBAAAAAL/RKXSL6ch0UrvNI3nqPuPw19aalDwM688LUdtYd922TQpawDH
viJKrZ2W4J00JAp+yjGkf/8QAGgEAAgMBAQAAAAAAAAAAAAAAAAAMBBAAUCBv/aAAGBAhAAAAAR2yP
O2tkxrv8S+P/EABkBAAlDAQAAAAAAAAAAAAAAAAAFAQQGA//aAAGBAxAAAAACb9ev01i5AbFIpGC8
//8QANBAAAgIBAgMFBgUEAwAAAAAAQIDBAUABgcREhMhIjFBFDJRYnKBCBAVQqFTYXGRI3Ox/9o
ACAEBAAE/ANHIW70skGFRCqErJdlHOFCPMRj95/jXEHi/w32HA4zmb13BmA5Q42pYDuJJz60jISI
D5tVvxt7auTOIOFkIh9Ge6Fcj7Ra4ecUtmb96Ku28xaw+Y6S36bccOj/QGJDj6DqrlJobEePy8Cw
WH7opUJME5+Ck+TfKfyyDyZG1HhKzsiFO1uSKerWInkIwfrn/APNceN0vsvhlxiXNa1cVMZS7Lw
1DP3Mw/uqAnUXC3dt5Ip6dSSwHTqPQpY6j2LuvFiT23b2TLwqSI46jv8AfuGrWA3ttaLG7nt46xT
AkVon6ws0bDxo3h5sp/zrhruJeJfDbb2dycQ7W5Ay2QP68DmIuvwJK8xrD2Z1exibrdVury/5P6s
Le5J/n0OsB0tWs5Jh47tmSQep7NT0IPsBr8RWIn/bG1JSomdbdGMef/rdyh1w8kQ0olji7h5nUzy
PA6dIOuL2MsWtp5/2eEuUgEpUeYER6iRr8LmThv8ACbFQIXvp2bUJT4EuX1nZVxtnHZn0jL15uXm
0cilh/plGtuOowOK+IrIDrfmHOC2jk6ar41aGwn1VpVmH8rrAVt9YaGfLpmIVhWszmvKhkhceagE
KvTrN4e/uGtUavlLtVexR2SvMU5u68+Z5FQwHwOptvLj6EyT2J7HbR9EomIIAI5eQ7tcFtvRbY2Z
QxUUaDs1DSunk8rgF21vdCNu2FHexli+56tYI+zm/iW7mqWHKA+sMx60P8kamj7WGaMnkHUr/ALG
s5kbuJ2xkaMkDA17TVJH8kVonAKuf2gj11j8hkMpiqlubHSU4Y41Q8uouXA5c1Ze4aFu1Yov7eJE
kUMCsg6SeXkdbf6IsFiliUL1VYj3D4oNZKFMhkaGJPNo0V7Vn16KAUQfcnWVq2Ip4Mxj4y9mBSks
Q854CeZuFMPNdVrte/VSzVk643+xBHmCPQjXGixgsPuPEVhN2VjORTpbjA8DCLpEc31gnlra/tQq
169mSvJFCp7Hkh6vCeRkK+Q1JZTK5T9KpsJrTP1OqnmEjT1b4AE6xOQgweMhx83azyIAkCjxPMxP
PpX7n7DWMpzVkmTxsGv2mEk5HkvosY+VB+VvDv2817FWjUuP7/hDRS/Wnx/uNcSeH1zcYpWc7EDL
UNns545AwME/SSPRkKlfdqngt/i3Xw0WYwtiWcq11wkK0UfyovIsx1szaX6NjWqYHHgCWTnNkrso
eawR++QLzJPwXuUaxuGgx7e1TObf4ry7Vx7oPogHuj8v/xAAjEQACAgEDAwUAAAAAAAAAAAAABAwI
RABAhMQQFYRIyQVFi/9oACAECAQE/AOMHUpn7WA4tsWi4SBHjQ7g4ulxkv1A2Tx5zthK5MT+r1mm
KmygRyLGdCqQLZgGtt9WwilsJz5oDL+BsPoaf/8QAIhEAAgEDAwUBAAAAAAAAAAAAAAQMCAAQREBI
hBRMiMwFi/9oACAEDAQE/AIgyIiPzo2D1kdxUqeISCBIEfDok7WrP6FXEJMYGY2j7XVYhioOzyAI
6puZXNpBwwDE811Rst2oGXkStt+a2125FuxK8beSMjkZo51IzkSZH2T70/9k=',
    },
    ],
    },
    ],
    },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        resourceInfo: 'resources',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    };
    this.toolbar = ['PdfExport'];
    this.splitterSettings = {
        columnIndex: 4,
    };
    this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName',
    };
    this.resources = editingResources;
}
public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttdefault_pdfexport') {
```

```

        let exportProperties: PdfExportProperties = {
            fileName: 'new.pdf',
        };
        this.ganttChart!.pdfExport(exportProperties);
    }
}
public pdfQueryCellInfo(args: any): void {
    if (args.column.headerText === 'Resources') {
        {
            args.image = {
                height: 40,
                width: 40,
                base64: (args as any).data.taskData.resourcesImage,
            };
        }
    }
    if (args.column.headerText === 'Email ID') {
        args.hyperLink = {
            target: 'mailto:' + (args as any).data.taskData.EmailId,
            displayText: (args as any).data.taskData.EmailId,
        };
    }
}
}
}

```

### MAIN.TS

```

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

```

### Exporting with taskbar template

The PDF export functionality allows to export taskbar templates that include images and text to an PDF document using [pdfQueryTaskbarInfo](#) event. Taskbars in the exported PDF document can be customized or formatted using the [pdfQueryTaskbarInfo](#) event for parent taskbar templates, taskbar templates and milestone templates.

In the following sample, taskbar templates with images and text are exported to PDF using [taskbarTemplate](#) properties in the [pdfQueryTaskbarInfo](#) event.

Note: PDF Export supports base64 string to export the images.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
    ToolbarService,
    PdfExportService,
    SelectionService,
} from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';

```

```

import {
  GanttComponent,
  ToolbarItem,
  PdfExportProperties
} from '@syncfusion/ej2-angular-gantt';
import { editingResources, base64Data } from './data';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
@Component({
  imports: [ GanttModule],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-gantt #ganttDefault id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [columns]="columns"
[toolbar]="toolbar" [rowHeight]="45" [taskbarHeight]="35"
(pdfQueryTaskbarInfo)="pdfQueryTaskbarInfo($event)"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'
[allowResizing] = 'true' [splitterSettings]="splitterSettings"
[resourceFields]="resourceFields" [resources]="resources"
>
  <ng-template #taskbarTemplate let-data>
  <div class="e-gantt-child-taskbar-inner-div e-gantt-child-taskbar"
style="position:absolute; height: 100%">
    <img [src]=" './' + data.ganttProperties.resourceInfo[0].resourceId +
'.png'" style="height: 40px; width: 40px;">
    <span class="e-span">{{ data.TaskName }}</span>
  </div>
  </ng-template>
  <ng-template #parentTemplate let-data>
  <div class="e-gantt-parent-taskbar-inner-div e-gantt-parent-taskbar"
style="position:absolute; height: 100%">
    <span class="e-span">{{ data.TaskName }}</span>
  </div>
  </ng-template>
  <ng-template #milestoneTemplate let-data>
  <div>
    <div class="e-gantt-milestone" style="position:absolute;">
      <div class="image" style="position:absolute; left: 8px ; top:
4px">
        <img class="moments"
src={{data.ganttProperties.resourceInfo[0].resourceId}}.png height="30px"
width="30px" />
      </div>
      <div
        class="e-milestone-top"
        style="border-right-width:26px; margin-top: -4px;border-
left-width:26px;border-bottom-width:26px;"
      ></div>
      <div
        class="e-milestone-bottom"
        style="top:16px;border-right-width:26px; border-left-
width:26px; border-top-width:26px;"
      ></div>
    </div>
  </div>
  </ng-template>

```

```

    </ejs-gantt>`,
    styleUrls: ['app.component.css'],
    encapsulation: ViewEncapsulation.None,
  })
  export class AppComponent {
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public resources?: object[];
    public rowHeight?: number;
    public toolbar?: ToolbarItem[];
    @ViewChild('ganttdefault', { static: true })
    public ganttChart?: GanttComponent;
    columns: ({ field: string; headerText: string; textAlign: string; width:
string; visible?: undefined; } | { field: string; headerText: string; width:
string; visible: boolean; textAlign?: undefined; } | { field: string;
headerText: string; width: string; textAlign?: undefined; visible?:
undefined; })[] | undefined;
    public resourceFields?: object;
    public ngOnInit(): void {
      this.data = base64Data,
      this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        resourceInfo: 'resources',
        startDate: 'StartDate',
        duration: 'Duration',
        child: 'subtasks',
      };
      this.toolbar = ['PdfExport'];
      this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left', width:
'100' },
        { field: 'TaskName', headerText: 'Task Name', width: '150' },
      ];
      this.splitterSettings = {
        columnIndex: 1,
      };
      this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName',
      };
      this.resources = editingResources;
    }
    public toolbarClick(args: ClickEventArgs): void {
      if (args.item.id === 'ganttdefault_pdfexport') {
        let exportProperties: PdfExportProperties = {
          fileName: 'new.pdf',
        };
        this.ganttChart!.pdfExport(exportProperties);
      }
    }
    public pdfQueryTaskbarInfo(args: any): void {
      if (!args.data.hasChildRecords) {
        if (args.data.ganttProperties.resourceNames) {
          args.taskbarTemplate.image = [{

```

```

        width: 20, base64: (args as any).data.taskData.resourcesImage,
height: 20
    ]]
    }
    args.taskbarTemplate.value = args.data.TaskName;
    }
    if (args.data.hasChildRecords) {
        if (args.data.ganttProperties.resourceNames) {
            args.taskbarTemplate.image = [{
                width: 20, base64: (args as any).data.taskData.resourcesImage,
height: 20
            ]]
        }
        args.taskbarTemplate.value = args.data.TaskName;
    }
    if (args.data.ganttProperties.duration === 0) {
        if (args.data.ganttProperties.resourceNames) {
            args.taskbarTemplate.image = [{
                width: 20, base64: (args as any).data.taskData.resourcesImage,
height: 20,
            ]]
        }
        args.taskbarTemplate.value = args.data.TaskName
    }
    }
}
}

```

## MAIN.TS

```

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

```

### Exporting with task label template

The PDF export functionality allows to export task label template that include images and text to an PDF document using [pdfQueryTaskbarInfo](#) event.

In the following sample, task label template with images and text are exported to PDF using [labelSettings](#) properties in the [pdfQueryTaskbarInfo](#) event.

Note: PDF Export supports base64 string to export the images.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
    ToolbarService,
    PdfExportService,
    SelectionService,
} from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import {
    GanttComponent,

```

```

    ToolbarItem,
    PdfExportProperties,
} from '@syncfusion/ej2-angular-gantt';
import { base64Data, editingResources } from './data';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
@Component({
  imports: [ GanttModule],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-gantt #ganttDefault id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
[labelSettings]="labelSettings"
(pdfQueryTaskbarInfo)="pdfQueryTaskbarInfo($event)"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'
[allowResizing] = 'true' [projectStartDate]="projectStartDate"
[projectEndDate]="projectEndDate" [splitterSettings]="splitterSettings"
[resourceFields]="resourceFields" [resources]="resources">
    <e-columns>
        <e-column field='TaskID' headerText='Task ID'
textAlign='Left'></e-column>
        <e-column field='TaskName' headerText='Task Name'
width='250'></e-column>

    </e-columns>
    <ng-template #labelSettingsLeftLabel let-data>
        <span>{{ data.TaskName }} [ {{ data.Progress }}% ]</span>
    </ng-template>
    <ng-template #labelSettingsRightLabel let-data>
        <div
            style="display:inline-flex"
            innerHtml="{{ customFunction(data) }}"></div>
    </ng-template>
</ejs-gantt>`,
  styleUrls: ['app.component.css'],
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  public labelSettings?: object;
  public resources?: object[];
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  public toolbar?: ToolbarItem[];
  public customFunction(data: any): string {
    var container = document.createElement('div');
    if (data.ganttProperties.resourceNames) {
      var resources = data.resources.split(',');
      for (var i = 0; i < resources.length; i++) {
        var subContainer = document.createElement('div');
        var img = document.createElement('img');
        var span = document.createElement('span');
        span.className = 'labelClass';
        span.innerHTML = resources[i];

```



```

        img.src =
'https://ej2.syncfusion.com/angular/demos/assets/gantt/images/' +
        resources[i] + '.png';
        img.height = 30;
        img.width = 30;
        subContainer.append(img);
        subContainer.append(span);
        container.append(subContainer);
    }
}
return container.innerHTML;
}
@ViewChild('ganttdefault', { static: true })
public ganttChart?: GanttComponent;

public resourceFields?: object;
public ngOnInit(): void {
    this.data = base64Data;
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        resourceInfo: 'resources',
        progress : 'Progress',
        startDate: 'StartDate',
        duration: 'Duration',
        child: 'subtasks',
    };
    this.toolbar = ['PdfExport'];
    this.splitterSettings = {
        columnIndex: 1,
    };
    this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName',
    };
    this.labelSettings = {
        taskLabel: '${Progress}%',
    };
    this.resources = editingResources;
    this.projectStartDate= new Date('03/24/2019');
    this.projectEndDate= new Date('04/30/2019');
}
public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttdefault_pdfexport') {
        let exportProperties: PdfExportProperties = {
            fileName: 'new.pdf',
        };
        this.ganttChart!.pdfExport(exportProperties);
    }
}
public pdfQueryTaskbarInfo(args: any): void {
    args.labelSettings.leftLabel.value = args.data.ganttProperties.taskName
+ '[' + args.data.ganttProperties.progress + '>';
    if (args.data.ganttProperties.resourceNames) {
        args.labelSettings.rightLabel.value =
args.data.ganttProperties.resourceNames;
        args.labelSettings.rightLabel.image = [{

```

```

        base64: (args as any).data.taskData.resourcesImage, width: 20,
height: 20
    ]]
    }
  }
}

```

## MAIN.TS

```

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

```

### Exporting with header template

The PDF export functionality allows to export header template that include images and text to an PDF document using [pdfColumnHeaderQueryCellInfo](#) event.

In the following sample, header template with images and text are exported to PDF using [headerTemplate](#) properties in the [pdfColumnHeaderQueryCellInfo](#) event.

Note: PDF Export supports base64 string to export the images.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import {
  ToolbarService,
  PdfExportService,
  SelectionService,
} from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import {
  GanttComponent,
  ToolbarItem,
  PdfExportProperties
} from '@syncfusion/ej2-angular-gantt';
import { editingResources, base64Data } from './data';
import { ClickEventArgs } from '@syncfusion/ej2-
navigations/src/toolbar/toolbar';
@Component({
  imports: [ GanttModule],
  providers: [ToolbarService, PdfExportService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-gantt #ganttDefault id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings" [toolbar]="toolbar"
[labelSettings]="labelSettings"
(pdfColumnHeaderQueryCellInfo)="pdfColumnHeaderQueryCellInfo($event)"
(toolbarClick)="toolbarClick($event)" allowPdfExport='true'
[allowResizing] = 'true' [projectStartDate]="projectStartDate"
[projectEndDate]="projectEndDate" [splitterSettings]="splitterSettings"
[resourceFields]="resourceFields" [resources]="resources">

```

```

        <e-columns>
            <e-column field='TaskName' width='250'>
                <ng-template #headerTemplate>
                     Task Name
                </ng-template>
            </e-column>
            <e-column field='StartDate'>
                <ng-template #headerTemplate>
                     Start Date
                </ng-template>
            </e-column>
        </e-columns>
    </ejs-gantt>`,
    styleUrls: ['app.component.css'],
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    public data?: object[];
    public taskSettings?: object;
    public splitterSettings?: object;
    public labelSettings?: object;
    public resources?: object[];
    public i: number = 0;
    public projectStartDate?: Date;
    public projectEndDate?: Date;
    public toolbar?: ToolbarItem[];
    @ViewChild('ganttDefault', { static: true })
    public ganttChart?: GanttComponent;

    public resourceFields?: object;
    public ngOnInit(): void {
        this.data = base64Data;
        this.taskSettings = {
            id: 'TaskID',
            name: 'TaskName',
            resourceInfo: 'resources',
            progress: 'Progress',
            startDate: 'StartDate',
            duration: 'Duration',
            child: 'subtasks',
        };
        this.toolbar = ['PdfExport'];
        this.splitterSettings = {
            columnIndex: 1,
        };
        this.resourceFields = {
            id: 'resourceId',
            name: 'resourceName',
        };
    }
}

```

```

    this.labelSettings = {
        taskLabel: '${Progress}%',
    };
    this.resources = editingResources;
    this.projectStartDate= new Date('03/24/2019');
    this.projectEndDate= new Date('04/30/2019');
}
public toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'ganttdefault_pdfexport') {
        let exportProperties: PdfExportProperties = {
            fileName: 'new.pdf',
        };
        this.ganttChart!.pdfExport(exportProperties);
    }
}
public pdfColumnHeaderQueryCellInfo(args: any): void {
    let base64Array: Object[] = [
        { 'TaskName':
            '/9j/4AAQSkZJRgABAQIAHAACAAD/4QBiRXhpZgAATU0AKgAAAAGABQESAAMAAABAAEAAAEaAAU
            AAAABAAAASgEbAAUAAAABAAAUgEoAAMAAABAAAMAAAIITAAMAAABAAEAAAAAAAAAAAcAAAAQA
            AABwAAAAB/9sAQwADAgICAgIDAgICAwMDAwQGBAQEBAQIBgYFBGkICgoJCAkJCgWPDAoLDgsJCQ0
            RDQ4PEBAREAoMEhmMSEBMPEBAQ/9sAQwEDAwMEAwQIBAQIEAsJCxAQEBAQEBAQEBAQEBAQEBA
            QEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQ/8AAEQgAIAAgAwERAAIRAQMRAf/EABGAAQE
            BAQEAAAAAAAAAAAAAAAAAYIAAcF/8QALBAAQQAQAgEDAwIHAAAAAAAAAAQIDBAUGBxEsAAgTIRQVQRY
            xFzhXdpao1f/EABQBAQAAAAAAAAAAAAAAAAAAD/xAAUEQEAAAAAAAAAAAAAAAAAA/9oADAM
            BAAIRAxEAPwB7gessOlaiw2zpdS4Ld2cqngOyl2rLbHcqjpliy6IzylL7/gp/J+RxwQQt68w6mew
            u7XrfEKC+azXGuiqiO2r2ybyqKnhD3stLVy2TyOg/cj5A5IXr4G8Cf9+aD0XT6K2Nb1GlsEgz4OJW
            8mLKjY5DaeYdRDdUhx0thSVJUAQoEEEAjwNNW2XoFprGLb1E/QEGdBerJiyoztK08w6hQUhx0kF
            KkqAIUCCCAR4CDD9sbV2RWSsol9r3BrDGza2NfWWEnOH21T2Yst2MJKUs1ryAhwslSeHffBHyRwS
            HnW26tv12qpO5Ier8GtMdYoVZI2qJm01L0iCGPfc0IeqEcKLfyErKT+DwfjwFvqO/162h/Zl3/ov
            eB0TWjTe2FRYxx5RqrLrj065HUuZRdzXIOQ7GRHc6yLV+YlmVDcgPJS6044AQVHhTY/I58Ao3lmJ
            UeibfRWBZH6bKCFbUL1K7PTtRpTrzjsQRlZJCWqxoPyFISkqWepUQOfj48Ctdj4j/ABA15lGB/cP
            oPlJSzaj6v2vd+n+oYW17nTsnv1789ew5445H7+Ad+x+oX+qGu/8AA53/AGPA5drHb+D4rru/xSy
            3nrPG86i5hkwnOXDjbTikG9lrU4qCqY271W0R0BfJSFI5UvqQQKWW5cOT6NMhxTZO+9d5F172ByI
            YjQrmM9LMo1oQll0iXIMuSH+3Z9BSlaiFBCeOSH//2Q==',
            { 'StartDate':
                '/9j/4AAQSkZJRgABAQIAHAACAAD/4QBiRXhpZgAATU0AKgAAAAGABQESAAMAAABAAEAAAEaAAU
                AAAABAAAASgEbAAUAAAABAAAUgEoAAMAAABAAAMAAAIITAAMAAABAAEAAAAAAAAAAAcAAAAQA
                AABwAAAAB/9sAQwADAgICAgIDAgICAwMDAwQGBAQEBAQIBgYFBGkICgoJCAkJCgWPDAoLDgsJCQ0
                RDQ4PEBAREAoMEhmMSEBMPEBAQ/9sAQwEDAwMEAwQIBAQIEAsJCxAQEBAQEBAQEBAQEBAQEBA
                QEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQEBAQ/8AAEQgAIAAgAwERAAIRAQMRAf/EABCAAQE
                BAQAAAAAAAAAAAAAAAAACABgX/xAAzEAAABAQDBwEGBwAAAAAAAAAABAgMEBQYHEQgSEwAUFRYYITI
                0IiQxMzVCN0NRVWwCg//EABUBAQEAAAAAAAAAAAAAAAAAAB/8QAGBEBAQEBAQAAAAAAAAAAAAA
                AABEBIUH/2gAMAwEAAhEDEQA/AG2t2PafKp4qHFI3sLlRGR4bE4QLEIi4Yu1XqLJdBsqs5UAU1spj
                EKqoJQBIfEoZTD8QCJcxtTdhwp3JlI6RxQ5yYQmGOVYjEVYbE8oPVoi8VFNmVRanEoInbjcUvIx
                wAw27BTHjim7EfTuc6R1ciEhyawi0MbKw6IpQ2J5ReoxFmqCagpC6OBRI4G4JeRSAJgv3B3ojj2
                nysGKhvSNlC5UWkeJROLpQ+It2LtJ6syQQcqt1BFRbKUxypJiYBSDyMGUo/AOtP7GoFVcRtTZRkW
                jGHI1Lcm8F3qKtVli68Qd72wIoTMs1m1MmmcgXAtigmAXsO1lSwYwJKqEwV0mLD8yw54TiTFLMNJF
                Xbl1WUHAMjpHK2MAJnC5xNZ2n2EgB2N37BdCqOpVQl+uku4fnuHPCceYpmhp4q0cpSg4FkRIhXJhB
                Q420UBrNFOWEE05e/cbIUuSAXqBSrEbTKUZ6oxh4hPOXGt1ikkS4uhEGm6MDqHyrK5dPPqEINgNco
                qANrhskLWameB0/jWL2uPPWiuYaV6PLO68Jm5CB8SvCy58+qA62nYlreOqN/INmGiCT5cpetjBnm
                EvcV00w2XUIAio0ndKem6L2Jq5GN2ykQEMixQEYgaYBcN3KH5Y7PTxThLlL0cYMjQlliummJS6vA
                FlHc7qz03WewxXI+s2TiABkRKIlTDTELjvBg/MDZ6eF+WIHT+C4vaHci4i5hqprczblxabkI5w20
                LNkyaQB06lZ3v5aQW8R2aYz1VOkrq9rPlSfx3gXlP9rJvPof8PP8Ar92zDQxLHQ71NzbzJ+EBHku
                AfV/X5Gefw968t8+Z7P6fZs4dUz9DvU3KXLf4QcGV4/8AV/X5HmTz968tz+X7P6/fs4dM9K+krq9
                ox0t/yLjv1P8Aaz7t67/fw/t9uzTH/9k=' },
    ]
}

```

```

while (this.i < base64Array.length) {
  const key = Object.keys(base64Array[this.i])[0];
  const value = (base64Array[this.i] as any)[key];
  if (key === args.column.field) {
    args.headerTemplate.image = [{
      base64: value, width: 20, height: 20
    }];
    args.headerTemplate.value = args.column.field;
    break;
  }
  this.i++;
}
}
}

```

### MAIN.TS

```

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

```

### Context menu in Angular Gantt component

The Gantt control allows you to perform quick actions by using context menu. When right-clicking the context menu, the context menu options are shown. To enable this feature, set the [enableContextMenu](#) to true. The default context menu options are enabled using the [editSettings](#) property. The context menu options can be customized using the [contextMenuItems](#) property.

To use the context menu, inject the `ContextMenuService` in the provider section of `AppModule`.

The default items are listed in the following table.

Items	Description
<code>AutoFit</code>	Auto-fits the current column.
<code>AutoFitAll</code>	Auto-fits all columns.
<code>SortAscending</code>	Sorts the current column in ascending order.
<code>SortDescending</code>	Sorts the current column in descending order.
<code>TaskInformation</code>	Edits the current task.
<code>Add</code>	Adds a new row to the Gantt.
<code>Indent</code>	Indent the selected record to one level.
<code>Outdent</code>	Outdent the selected record to one level.
<code>DeleteTask</code>	Deletes the current task.
<code>Save</code>	Saves the edited task.
<code>Cancel</code>	Cancels the edited task.
<code>DeleteDependency</code>	Deletes the current dependency task link.

**Convert** | Converts current task to milestone or vice-versa.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule, ContextMenuService, EditService, SortService,
ResizeService } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService, ContextMenuService, EditService, SortService,
  ResizeService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttContextmenu" height="430px"
[dataSource]="editingData" [taskFields]="taskSettings"
[enableContextMenu]="true" [allowSorting]="true" [allowResizing]="true"
[editSettings]="editSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public editingData?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public ngOnInit(): void {
    this.editingData = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.editSettings = {
      allowAdding: true,
      allowEditing: true,
      allowDeleting: true
    };
  }
}
```

### MAIN.TS

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

### Custom context menu items

The custom context menu items can be added by defining the [contextMenuItems](#) as a collection of [contextMenuItemModel](#).

Actions for the customized items can be defined in the [contextMenuClick](#) event and the visibility of customized items can be defined in the [contextMenuOpen](#) event.

To create custom context menu items for header area, define the target property as `.e-gridheader`.

The following sample shows context menu item for parent rows to expand or collapse child rows in the content area and a context menu item to hide columns in the header area.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule, SortService, ResizeService } from '@syncfusion/ej2-angular-gantt'
import { SelectionService, ContextMenuService, EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent, ContextMenuClickEventArgs, ContextMenuOpenEventArgs } from '@syncfusion/ej2-angular-gantt';
import { ContextMenuItemModel } from '@syncfusion/ej2-grids';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService, ContextMenuService, EditService, SortService, ResizeService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #customcontextmenu id="ganttCustomContextMenu"
height="430px" [dataSource]="editingData" [taskFields]="taskSettings"
[enableContextMenu]="true" [contextMenuItems]="contextMenuItems"
[allowSorting]="true" [allowResizing]="true" [editSettings]="editSettings"
(contextMenuClick)="contextMenuClick($event)"
(contextMenuOpen)="contextMenuOpen($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public editingData?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public contextMenuItems?: (string | ContextMenuItemModel)[];
  @ViewChild('customcontextmenu', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.editingData = editingData;
    this.taskSettings = {
```

```

        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.editSettings = {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true
    };
    this.contextMenuItems = ['AutoFitAll', 'AutoFit', 'TaskInformation',
        'DeleteTask', 'Save', 'Cancel', 'SortAscending', 'SortDescending', 'Add',
        'DeleteDependency', 'Convert',
        { text: 'Collapse the Row', target: '.e-content', id: 'collapserow'
    } as ContextMenuItemModel,
        { text: 'Expand the Row', target: '.e-content', id: 'expandrow' } as
ContextMenuItemModel,
        { text: 'Hide Column', target: '.e-gridheader', id: 'hidecols' } as
ContextMenuItemModel
    ];
    }
    public contextMenuClick (args: ContextMenuClickEventArgs) {
        let record = args.rowData;
        if (args.item.id === 'collapserow') {
            this.ganttObj.collapseByID(Number(record!.ganttProperties!.taskId));
        }
        if (args.item.id === 'expandrow') {
            this.ganttObj.expandByID(Number(record!.ganttProperties!.taskId));
        }
        if (args.item.id === 'hidecols') {
            this.ganttObj.hideColumn(args.column!.headerText);
        }
    }
    public contextMenuOpen (args: ContextMenuOpenEventArgs) {
        let record = args.rowData;
        if (args.type !== 'Header') {
            if (!record!.hasChildRecords) {
                args.hideItems!.push('Collapse the Row');
                args.hideItems!.push('Expand the Row');
            } else {
                if (record!.expanded) {
                    args.hideItems!.push("Expand the Row");
                } else {
                    args.hideItems!.push("Collapse the Row");
                }
            }
        }
    }
}

```

**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 show an specific item in context menu for header/content area in the Gantt control by defining the `target` property.

#### Touch interaction

To perform `long press` action on a row, [context menu](#) is opened, and then tap a menu item to trigger its action.

#### State persistence in Angular Gantt component

State persistence refers to the Gantt's state maintained in the browser's [localStorage](#) even if the browser is refreshed or if you move to the next page within the browser.

State persistence stores gantt's model object in the local storage when the [enablePersistence](#) is defined as true.

#### Get or set localStorage value

If the [enablePersistence](#) property is set to true, the gantt property value is saved in the `window.localStorage` for reference. You can get/set the localStorage value by using the `getItem/setItem` method in the `window.localStorage`.

`typescript

//get the Gantt model.

let value: string = window.localStorage.getItem('ganttgantt'); //"ganttgantt" is component name + component id.

let model: Object = JSON.parse(model);

,

`typescript

//set the Gantt model.

window.localStorage.setItem('ganttgantt', JSON.stringify(model)); //"ganttgantt" is component name + component id.

,

You can refer to our [Angular Gantt](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Gantt example](#) to know how to present and manipulate data.

#### Prevent columns from persisting

When the [enablePersistence](#) property is set to true, the Gantt properties such as [Filtering](#), [Sorting](#), and [Columns](#) will persist. You can use the `addOnPersist` method to prevent these Gantt properties from persisting.

The following example demonstrates how to prevent Gantt columns from persisting. In the [dataBound](#) event of the Gantt, you can override the `addOnPersist` method and remove the columns from the key list given for persistence.

Note: When the `enablePersistence` property is set to true, the Gantt features such as column template, column formatter, header text, and value accessor will not persist.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { CriticalPathService, ToolbarService, EditService } from '@syncfusion/ej2-angular-gantt'
import { Component, OnInit, ViewChild, ViewEncapsulation } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button id='add' (click)='addColumn()'>Add
Columns</button>
      <button ej-button id='remove' (click)='removeColumn()'>Remove
Columns</button>
      <ejs-gantt id="ganttDefault" #gantt height="430px"
[dataSource]="data" [taskFields]="taskSettings" enablePersistence='true'
[columns]="columns" (dataBound)='dataBound()'></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent implements OnInit {
  public data?: object[];
  public taskSettings?: object;
  public ganttChart?: GanttComponent;
  public splitterSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
        ]
      },
      {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
```

```

        { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
        { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
    ]
    },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    };
    this.columns = [
120 { field: 'TaskID', headerText: 'Task ID', textAlign: 'Right', width:
    { field: 'TaskName', headerText: 'Task Name', width: 150},
    { field: 'StartDate', headerText: 'StartDate', width: 150 },
    { field: 'Duration', headerText: 'Duration', width: 150},
    { field: 'Progress', headerText: 'Progress', width: 150 },
    ];
    }
    dataBound() {
        let gantt = (document.getElementsByClassName('e-gantt')[0] as
any).ej2_instances[0];
        let cloned = gantt.addOnPersist;
        gantt.addOnPersist = function (key: any) {
            key = key.filter((item: string) => item !== "columns");
            return cloned.call(this, key);
        };
    }
    addColumn() {
        let gantt = (document.getElementsByClassName('e-gantt')[0] as
any).ej2_instances[0];
        let obj = { field: "Progress", headerText: 'Progress', width: 100 };
        gantt.columns.push(obj as any); //you can add the columns by using
the Gantt columns method
        gantt.refresh();
    }
    removeColumn() {
        let gantt = (document.getElementsByClassName('e-gantt')[0] as
any).ej2_instances[0];
        gantt.columns.pop();
        gantt.refresh();
    }
}
}

```

## MAIN.TS

```

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

```

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

### Persist the header template and header Text

By default, the Gantt properties such as column template, header text, header template, column formatter, and value accessor will not persist when [enablePersistence](#) is set to true. Because the column template and header text can be customized at the application level.

If you wish to restore all these column properties, then you can achieve it by cloning the gantt columns property using JavaScript Object's assign method and storing this along with the persist data manually. While restoring the settings, this column object must be assigned to the gantt's columns property to restore the column settings as demonstrated in the following sample.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { CriticalPathService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, OnInit, ViewChild, ViewEncapsulation } from
 '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button id='restore'
(click)= 'clickHandler()'>Restore</button>
<ejs-gantt id="ganttDefault" #gantt height="430px"
[dataSource]="data" [taskFields]="taskSettings" enablePersistence='true'
[columns]="columns" [splitterSettings] = "splitterSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent implements OnInit {
  public data?: object[];
  public taskSettings?: object;
  public ganttChart?: GanttComponent;
  public splitterSettings?: object;
  public columns?: object[];
  public ngOnInit(): void {
    this.data = [
      {
        TaskID: 1,
        TaskName: 'Project Initiation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
          { TaskID: 2, TaskName: 'Identify Site location',
StartDate: new Date('04/02/2019'), Duration: 4, Progress: 50 },
          { TaskID: 3, TaskName: 'Perform Soil test', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
```

```

        { TaskID: 4, TaskName: 'Soil test approval', StartDate:
new Date('04/02/2019'), Duration: 4, Progress: 50 },
    ],
    },
    {
        TaskID: 5,
        TaskName: 'Project Estimation',
        StartDate: new Date('04/02/2019'),
        EndDate: new Date('04/21/2019'),
        subtasks: [
            { TaskID: 6, TaskName: 'Develop floor plan for
estimation', StartDate: new Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 7, TaskName: 'List materials', StartDate: new
Date('04/04/2019'), Duration: 3, Progress: 50 },
            { TaskID: 8, TaskName: 'Estimation approval', StartDate:
new Date('04/04/2019'), Duration: 3, Progress: 50 }
        ]
    },
    ];
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        duration: 'Duration',
        progress: 'Progress',
        child: 'subtasks',
    };
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Right', width:
120 },
        { field: 'TaskName', headerText: 'Task Name', width: 150,
headerTemplate: '#customertemplate' },
        { field: 'StartDate', headerText: 'StartDate', width: 150 },
        { field: 'Duration', headerText: 'Duration', width: 150 },
        { field: 'Progress', headerText: 'Progress', width: 150 },
    ];
}
clickHandler() {
    let gantt = (document.getElementsByClassName('e-gantt')[0] as
any).ej2_instances[0];
    var savedProperties = JSON.parse(gantt.getPersistData());
    var gridColumnsState = Object.assign([], gantt.ganttColumns);
    savedProperties.columns.forEach(function (col: { field: any;
headerText: string; template: any; headerTemplate: any; }) {
        var headerText = gridColumnsState.find(function
(colColumnsState: { field: any; }) { return colColumnsState.field ===
col.field; })['headerText'];
        var colTemplate = gridColumnsState.find(function
(colColumnsState: { field: any; }) { return colColumnsState.field ===
col.field; })['template'];
        var headerTemplate = gridColumnsState.find(function
(colColumnsState: { field: any; }) { return colColumnsState.field ===
col.field; })['headerTemplate'];
        col.headerText = 'Text Changed';
        col.template = colTemplate;
        col.headerTemplate = headerTemplate;
    });
}

```

```

        console.log(savedProperties);
        gantt.treeGrid.setProperties(savedProperties);
    }
}

```

### MAIN.TS

```

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

```

### Timezone in Angular Gantt component

The Gantt makes use of the current system time zone by default. If it needs to follow some other user-specific time zone, then the `timezone` property needs to be used.

#### Understanding date manipulation in JavaScript

The `new Date()` in JavaScript returns the exact current date object with complete time and timezone information. For example, it may return value such as `Wed Dec 12 2018 05:23:27 GMT+0530 (India Standard Time)` which indicates that the current date is December 12, 2018 and the current time is 5.23 AM on browsers following the IST timezone.

#### Display same time everywhere with no time difference

Setting `timezone` to UTC for Gantt will display the same time as in the database for all the users in different time zone.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { Gantt } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt [dataSource]="data"
    [taskFields]="taskSettings" [dayWorkingTime]="dayWorkingTime"
    [timelineSettings]="timelineSettings" timezone="UTC" durationUnit="Hour"
    dateFormat="hh:mm a" height="450px" [includeWeekend]="true">`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  public ganttObj?: GanttComponent | any;
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;

```

```

    public timezoneValue: string = 'UTC';
    public dayWorkingTime?: object[];
    public ngOnInit(): void {
        this.data = [
            { taskID: 1, taskName: 'Project Schedule', startDate: new
Date('02/04/2019 08:00'), endDate: new Date('03/10/2019') },
            { taskID: 2, taskName: 'Planning', startDate: new
Date('02/04/2019 08:00'), endDate: new Date('02/10/2019'), parentID: 1 },
            { taskID: 3, taskName: 'Plan timeline', startDate: new
Date('02/04/2019 08:00'), endDate: new Date('02/10/2019'), duration: 6,
progress: '60', parentID: 2 },
            { taskID: 4, taskName: 'Plan budget', startDate: new
Date('02/04/2019 08:00'), endDate: new Date('02/10/2019'), duration: 6,
progress: '90', parentID: 2 },
            { taskID: 5, taskName: 'Allocate resources', startDate: new
Date('02/04/2019 08:00'), endDate: new Date('02/10/2019'), duration: 6,
progress: '75', parentID: 2 },
            { taskID: 6, taskName: 'Planning complete', startDate: new
Date('02/06/2019 08:00'), endDate: new Date('02/10/2019'), duration: 0,
predecessor: '3FS,4FS,5FS', parentID: 2 },
            { taskID: 7, taskName: 'Design', startDate: new Date('02/13/2019
08:00'), endDate: new Date('02/17/2019 08:00'), parentID: 1 },
            { taskID: 8, taskName: 'Software Specification', startDate: new
Date('02/13/2019 08:00'), endDate: new Date('02/15/2019'), duration: 3,
progress: '60', predecessor: '6FS', parentID: 7 },
            { taskID: 9, taskName: 'Develop prototype', startDate: new
Date('02/13/2019 08:00'), endDate: new Date('02/15/2019'), duration: 3,
progress: '100', predecessor: '6FS', parentID: 7 },
            { taskID: 10, taskName: 'Get approval from customer', startDate:
new Date('02/16/2019 08:00'), endDate: new Date('02/17/2019 08:00'),
duration: 2, progress: '100', predecessor: '9FS', parentID: 7 },
            { taskID: 11, taskName: 'Design complete', startDate: new
Date('02/17/2019 08:00'), endDate: new Date('02/17/2019 08:00'), duration:
0, predecessor: '10FS', parentID: 7 }
        ];
        this.taskSettings = {
            id: 'taskID',
            name: 'taskName',
            startDate: 'startDate',
            duration: 'duration',
            progress: 'progress',
            dependency: 'predecessor',
            parentID: 'parentID'
        };
        this.timelineSettings = {
            timelineUnitSize: 65,
            topTier: {
                unit: 'Day',
                format: 'MMM dd, yyyy'
            },
            bottomTier: {
                unit: 'Hour',
                format: 'hh:mm a'
            }
        };
        this.dayWorkingTime = [{ from: 0, to: 24 }];
    }

```

```
}

```

### MAIN.TS

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

```

### CRUD operations with timezone

CRUD operations can be performed with timezone, and the gantt is rendered based on the timezone specified in the load time. All the editing actions will be done based on the user timezone, but on database save action, we have reversed this conversion to local time and provided data to client side events for the better understanding. Refer to the following code example.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" #gantt [dataSource]="data"
    [taskFields]="taskSettings" [dayWorkingTime]="dayWorkingTime"
    (actionComplete)="actionComplete($event)"
    [timelineSettings]="timelineSettings" timezone="America/New_York"
    durationUnit="Hour" dateFormat="hh:mm a" height="450px"
    [includeWeekend]="true">`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  public ganttObj?: GanttComponent;
  public data?: object[];
  public taskSettings?: object;
  public timelineSettings?: object;
  public timezoneValue: string = 'UTC';
  public dayWorkingTime?: object[];
  public editSettings?: object;
  public ngOnInit(): void {
    this.data = [
      {
        taskID: 1,
        taskName: 'Project Schedule',
        startDate: new Date('02/04/2019 08:00'),
        endDate: new Date('03/10/2019')
      }
    ]
  }
}
```



```

    },
    {
      taskID: 2,
      taskName: 'Planning',
      startDate: new Date('02/04/2019 08:00'),
      endDate: new Date('02/10/2019'),
      parentID: 1
    },
    {
      taskID: 3,
      taskName: 'Plan timeline',
      startDate: new Date('02/04/2019 08:00'),
      endDate: new Date('02/10/2019'),
      duration: 6,
      progress: '60',
      parentID: 2
    },
    {
      taskID: 4,
      taskName: 'Plan budget',
      startDate: new Date('02/04/2019 08:00'),
      endDate: new Date('02/10/2019'),
      duration: 6,
      progress: '90',
      parentID: 2
    },
    {
      taskID: 5,
      taskName: 'Allocate resources',
      startDate: new Date('02/04/2019 08:00'),
      endDate: new Date('02/10/2019'),
      duration: 6,
      progress: '75',
      parentID: 2
    },
    {
      taskID: 6,
      taskName: 'Planning complete',
      startDate: new Date('02/06/2019 08:00'),
      endDate: new Date('02/10/2019'),
      duration: 0,
      predecessor: '3FS,4FS,5FS',
      parentID: 2
    },
    {
      taskID: 7,
      taskName: 'Design',
      startDate: new Date('02/13/2019 08:00'),
      endDate: new Date('02/17/2019 08:00'),
      parentID: 1
    },
    {
      taskID: 8,
      taskName: 'Software Specification',
      startDate: new Date('02/13/2019 08:00'),
      endDate: new Date('02/15/2019'),
      duration: 3,

```

```

        progress: '60',
        predecessor: '6FS',
        parentID: 7
    },
    {
        taskID: 9,
        taskName: 'Develop prototype',
        startDate: new Date('02/13/2019 08:00'),
        endDate: new Date('02/15/2019'),
        duration: 3,
        progress: '100',
        predecessor: '6FS',
        parentID: 7
    },
    {
        taskID: 10,
        taskName: 'Get approval from customer',
        startDate: new Date('02/16/2019 08:00'),
        endDate: new Date('02/17/2019 08:00'),
        duration: 2,
        progress: '100',
        predecessor: '9FS',
        parentID: 7
    },
    {
        taskID: 11,
        taskName: 'Design complete',
        startDate: new Date('02/17/2019 08:00'),
        endDate: new Date('02/17/2019 08:00'),
        duration: 0,
        predecessor: '10FS',
        parentID: 7
    }
];
this.taskSettings = {
    id: 'taskID',
    name: 'taskName',
    startDate: 'startDate',
    duration: 'duration',
    progress: 'progress',
    dependency: 'predecessor',
    parentID: 'parentID'
};
this.editSettings = {
    allowAdding: true,
    allowEditing: true,
    allowDeleting: true,
    allowTaskbarEditing: true,
    showDeleteConfirmDialog: true
};
this.timelineSettings = {
    timelineUnitSize: 65,
    topTier: {
        unit: 'Day',
        format: 'MMM dd, yyyy'
    },
    bottomTier: {

```

```

        unit: 'Hour',
        format: 'hh:mm a'
    }
};
this.dayWorkingTime = [{ from: 0, to: 24 }];
}
actionComplete(args: any) {
    if (args.action == "TaskbarEditing") {
        console.log(args.data.ganttProperties.endDate);
    }
}
}
}

```

## 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

`

[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

`

## Global local in Angular Gantt component

### Localization

The [Localization](#) library allows you to localize default text content of the Gantt.

The Gantt component has static text on some features (like column headers, add and edit dialog, tooltip, toolbar, etc.) that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

The following list of properties and its values are used in the Gantt.

Locale key words | Text

emptyRecord | No records to display

id | ID  
name | Name  
startDate | Start Date  
endDate | End Date  
duration | Duration  
progress | Progress  
dependency | Dependency  
notes | Notes  
baselineStartDate | Baseline Start Date  
baselineEndDate | Baseline End Date  
type | Type  
offset | Offset  
resourceName | Resources  
resourceID | Resource ID  
day | day  
hour | hour  
minute | minute  
days | days  
hours | hours  
minutes | minutes  
generalTab | General  
customTab | Custom Columns  
writeNotes | Write Notes  
addDialogTitle | New Task  
editDialogTitle | Task Information  
add | Add  
edit | Edit  
update | Update  
delete | Delete  
cancel | Cancel  
search | Search  
task | task  
tasks | tasks

zoomIn | Zoom in

zoomOut | Zoom out

zoomToFit | Zoom to fit

expandAll | Expand all

collapseAll | Collapse all

nextTimeSpan | Next timespan

prevTimeSpan | Previous timespan

saveButton | Save

taskBeforePredecessor\_FS | You moved "{0}" to start before "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_FS | You moved "{0}" away from "{1}" and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_SS | You moved "{0}" to start before "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_SS | You moved "{0}" to start after "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_FF | You moved "{0}" to finish before "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_FF | You moved "{0}" to finish after "{1}" finishes and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskBeforePredecessor\_SF | You moved "{0}" away from "{1}" to starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

taskAfterPredecessor\_SF | You moved "{0}" to finish after "{1}" starts and the two tasks are linked. As the result, the links cannot be honored. Select one action below to perform

okText | Ok

confirmDelete | Are you sure you want to Delete Record?

from | From

to | To

taskLink | Task Link

lag | Lag

start | Start

finish | Finish

enterValue | Enter the value

taskInformation | Task Information

deleteTask | Delete Task

deleteDependency | Delete Dependency

convert | Convert

save | Save

above | Above

below | Below

child | Child

milestone | Milestone

toTask | To Task

toMilestone | To Milestone

eventMarkers | Event markers

leftTaskLabel | Left task label

rightTaskLabel | Right task label

timelineCell | Timeline cell

confirmPredecessorDelete | Are you sure you want to remove dependency link?taskMode | Task Mode

changeScheduleMode | Change Schedule Mode

subTasksStartDate | SubTasks Start Date

subTasksEndDate | SubTasks End Date

scheduleStartDate | Schedule Start Date

scheduleEndDate | Schedule End Date

auto | Auto

manual | Manual

zoomToFit | Zoom to fit

excelExport | Excel export

csvExport | CSV export

pdfExport | Pdf export

unit | Unit

work | Work

taskType | Task Type

unassignedTask | Unassigned Task

group | Group

*Loading translations*

To load translation object in an application use [load](#) function of [L10n](#) class.

The below example demonstrates the Gantt in **Deutsch** culture.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
```

```

import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { Ganttdata } from './data';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'gantt': {
      "id": "Ich würde",
      "name": "Name",
      "startDate": "Anfangsdatum",
      "duration": "Dauer",
      "progress": "Fortschritt",
    }
  }
});
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" locale="de-DE"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    this.data = Ganttdata;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
  }
}

```

## MAIN.TS

```

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

```

## Internationalization

The [Internationalization](#) library is used to globalize number, date, and time values in gantt component.



**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { L10n, loadCldr, setCulture } from '@syncfusion/ej2-base';
import * as gregorian from './ca-gregorian.json';
import * as numbers from './numbers.json';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { Ganttdata } from './data';
L10n.load({
  'de-DE': {
    'gantt': {
      "id": "Ich würde",
      "name": "Name",
      "startDate": "Anfangsdatum",
      "duration": "Dauer",
      "progress": "Fortschritt",
    }
  }
});
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" locale="de-DE"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public ngOnInit(): void {
    setCulture('de-DE');
    loadCldr(
      './numbers.json',
      './ca-gregorian.json',
    );
    this.data = Ganttdata;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
  }
}

```

**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, **Timeline** is formatted by **NumberFormatOptions** and **DateFormatOptions**.

\* By default, **locale** value is **en-US**. If you want to change **en-US** culture, then set the **locale**.

### Right to left (RTL)

RTL provides an option to switch the text direction and layout of the Gantt component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Urdu, etc.). To enable RTL Gantt, set the **enableRtl** to true.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { L10n, loadCldr, setCulture } from '@syncfusion/ej2-base';
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt, ToolbarItem } from '@syncfusion/ej2-gantt';
import { Ganttdata } from './data';
setCulture('ar-AE');
L10n.load({
  'ar-AE': {
    "gantt": {
      "emptyRecord": "لا سجلات لعرضها",
      "id": "هوية شخصية",
      "name": "اسم",
      "startDate": "تاريخ البدء",
      "endDate": "تاريخ الانتهاء",
      "duration": "المدة الزمنية",
      "progress": "تقدم",
      "dependency": "الاعتماد",
      "notes": "ملاحظات",
      "baselineStartDate": "تاريخ البدء الأساسي",
      "baselineEndDate": "تاريخ نهاية خط الأساس",
      "taskMode": "وضع المهام",
      "changeScheduleMode": "تغيير وضع الجدول",
      "subTasksStartDate": "تاريخ بدء المهام الفرعية",
      "subTasksEndDate": "تاريخ انتهاء المهام الفرعية",
      "scheduleStartDate": "جدولة تاريخ البدء",
      "scheduleEndDate": "تاريخ انتهاء الجدول الزمني",
      "auto": "تلقائي",
      "manual": "كتيب",
      "type": "اكتب",
      "offset": "عوض",
      "resourceName": "مصادر",
      "resourceID": "معرف المورد",
      "day": "يوم",
      "hour": "ساعة",
      "minute": "دقيقة",
      "days": "أيام",
      "hours": "ساعات",
      "minutes": "الدقائق",
```

```

"generalTab": "جنرال لواء",
"customTab": "أعمدة مخصصة",
"writeNotes": "اكتب ملاحظات",
"addDialogTitle": "مهمة جديدة",
"editDialogTitle": "معلومات المهمة",
"saveButton": "حفظ",
"add": "إضافة",
"edit": "تعديل",
"update": "تحديث",
"delete": "حذف",
"cancel": "إلغاء",
"search": "بحث",
"task": "مهمة",
"tasks": "مهام",
"zoomIn": "تكبير",
"zoomOut": "تصغير",
"zoomToFit": "تكبير لتناسب",
"excelExport": "اكسل التصدير",
"csvExport": "تصدير CSV",
"expandAll": "توسيع الكل",
"collapseAll": "انهيار جميع",
"nextTimeSpan": "الجدول الزمني التالي",
"prevTimeSpan": "الجدول الزمني السابق",
"okText": "حسنًا",
"confirmDelete": "هل أنت متأكد أنك تريد حذف السجل؟",
"from": "من عند",
"to": "إلى",
"taskLink": "رابط المهمة",
"lag": "بطئ",
"start": "بداية",
"finish": "إنهاء",
"enterValue": "أدخل القيمة",
"taskBeforePredecessor_FS": "0 '{ قبل انتهاء' للبدء قبل انتهاء' و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء '{1}' واحدًا أدناه للقيام به",
"taskAfterPredecessor_FS": "0 '{ بعيدًا عن 1' '{ ويتم' و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدًا أدناه للقيام به",
"taskBeforePredecessor_SS": "0 '{ للبدء قبل أن يبدأ' للبدء قبل انتهاء' و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدًا '{1}' واحدًا أدناه للقيام به",
"taskAfterPredecessor_SS": "0 '{ للبدء بعد بدء 1' '{ و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدًا أدناه للقيام به",
"taskBeforePredecessor_FF": "0 '{ للإنهاء قبل انتهاء' للبدء بعد انتهاء' و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء '{1}' واحدًا أدناه للقيام به",
"taskAfterPredecessor_FF": "0 '{ للإنهاء بعد انتهاء' للبدء بعد انتهاء' و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء '{1}' واحدًا أدناه للقيام به",
"taskBeforePredecessor_SF": "0 '{ بعيدًا عن 1' '{ للبدء' والتشغيل وترتبط المهمتان. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدًا أدناه للقيام به",
"taskAfterPredecessor_SF": "0 '{ للإنهاء بعد بدء 1' '{ و يتم ربط المهمتين. ونتيجة لذلك ، لا يمكن احترام الروابط. حدد إجراء واحدًا أدناه للقيام به",
"taskInformation": "معلومات المهمة",

```

```

        "deleteTask": "حذف المهمة",
        "deleteDependency": "حذف التبعية",
        "convert": "تحويل",
        "save": "حفظ",
        "above": "في الاعلى",
        "below": "أدناه",
        "child": "طفل",
        "milestone": "معلما",
        "toTask": "المهمة",
        "toMilestone": "إلى معلم",
        "eventMarkers": "علامات الحدث",
        "leftTaskLabel": "تسمية المهمة اليسرى",
        "rightTaskLabel": "تسمية المهمة الصحيحة",
        "timelineCell": "خلية الجدول الزمني",
        "confirmPredecessorDelete": "هل أنت متأكد أنك تريد إزالة رابط",
        "التبعية؟",
        "unit": "وحدة",
        "work": "عمل",
        "taskType": "نوع المهمة",
        "unassignedTask": "مهمة غير محددة",
        "group": "مجموعة",
        "indent": "مسافة بادئة",
        "outdent": "عفا عليها الزمن",
        "segments": "شرائح",
        "splitTask": "تقسيم المهمة",
        "mergeTask": "مهمة الدمج",
        "left": "اليسار",
        "right": "حق"
    }
}
});
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" locale="ar-AE" [enableRtl]='true'
    [toolbar]="toolbar"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  public ngOnInit(): void {
    this.data = Ganttdata;
    this.toolbar = ['ExpandAll', 'CollapseAll'];
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',

```

```

        child: 'subtasks'
      };
    }
  }
}

```

## 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 Gantt component

The Gantt component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Gantt component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation | 

|

```

<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>

<div> - All
features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

```

### WAI-ARIA attributes

The Gantt component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Gantt component:

The following ARIA attributes are used in Gantt:

| Attributes | Purpose |

| --- | --- |

| **grid (role)** | This attribute is added to the **e-table** element present in the Gantt, which represents Grid part |

| **gridcell (role)** | This attribute is added to the **td** elements present within the **e-table**, which represents the work cells of Gantt |

| **columnheader (role)** | This attribute is added to the **th** elements within the **e-table**, which represents the header cells of Grid table |

| **separator (role)** | This attribute is added to the **e-split-bar** element, which represents the splitter between the Grid table and Chart |

| **dialog (role)** | This attribute is added to the **e-dialog** element, which represents the pop-up dialog |

| **toolbar (role)** | This attribute is added to the **e-gantt-toolbar** element, which represents the toolbars of Gantt |

| **aria-label** | It indicates the element's information<br> It is assigned to the Gantt UI elements such as timeline cell, taskbar, left label, right label, dependency line, and event markers. |

| **aria-selected** | This attribute is assigned to the Gantt chart row, and it defaults to **false**. The value is changed to **true** when the user selects a grid cell or task |

| **aria-expanded** | This attribute is assigned to the Gantt chart parent task row. The value is changed to **true** when the user clicks a parent taskbar to expand. After the user clicked a parent taskbar to collapse, the attribute value is changed to **false** |

| **aria-grabbed** | This attribute is assigned to the taskbars of Gantt when the user tries to achieve taskbar editing |

### Keyboard navigation

The Gantt component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Gantt component.

| **Press** | **To do this** |

| --- | --- |

| **Home** | Selects the first row. |

| **End** | Selects the last row. |

| **DownArrow** | Moves the cell focus/row or cell selection downward. |

| **UpArrow** | Moves the cell focus/row or cell selection upward. |

| **LeftArrow** | Moves the cell focus/row or cell selection left side. |

| **RightArrow** | Moves the cell focus/row or cell selection right side. |

| **Ctrl + Up Arrow** | Collapses all tasks. |

| **Ctrl + Down Arrow** | Expands all tasks. |

| **Ctrl + Shift + Up Arrow** | Collapses the selected row. |

| **Ctrl + Shift + Down Arrow** | Expands the selected row. |

| **Enter** | Saves request. |

| **Esc** | Cancels request. |

| **Insert** | Adds a new row. |

| **Ctrl + Insert** | Opens addRowDialog. |

| **Ctrl + F2** | Opens editRowDialog. |

| **Delete** | Deletes the selected row. |

| **Shift + F5** | FocusTask |

| **Ctrl + Shift + F** | Focus search |

| **Shift + DownArrow** | Extends the row/cell selection downwards. |

| **Shift + UpArrow** | Extends the row/cell selection upwards. |

| **Shift + LeftArrow** | Extends the cell selection to the left side. |

| **Shift + RightArrow** | Extends the cell selection to the right side. |

| **Tab / Shift + Tab** | To focus the close icon in the message. |

| **Alt + j** | Focus Gantt component. |

### Ensuring accessibility

The Gantt component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Gantt component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Gantt component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

### Ej1 api migration in Angular Gantt component

This topic shows the API equivalent of JS2 Gantt component to be used, while migrating your project that uses JS1 Gantt.

#### Data Binding and Task mapping

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Data Binding | **Property:** *dataSource* <br/><br/> <ej-gantt [dataSource]="ganttData"><br/></ej-gantt> | **Property:** *dataSource* <br/><br/> <ejs-gantt [dataSource]="ganttData"><br/></ejs-gantt> |

| To map id of task from data source | **Property:** *taskIdMapping* <br/><br/> <ej-gantt taskIdMapping="taskId"><br/></ej-gantt> | **Property:** *taskFields.id* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { id: 'TaskID' }; |

| To map name of task from data source | **Property:** *taskNameMapping* <br/><br/> <ej-gantt taskNameMapping="taskName"><br/></ej-gantt> | **Property:** *taskFields.name* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { name: 'TaskName' }; |

| To map start date from data source | **Property:** *startDateMapping* <br/><br/> <ej-gantt startDateMapping="startDate"><br/></ej-gantt> | **Property:** *taskFields.startDate* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { startDate: 'StartDate' }; |

| To map end date from data source | **Property:** *endDateMapping* <br/><br/> <ej-gantt endDateMapping="EndDate"><br/></ej-gantt> | **Property:** *taskFields.endDate* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { endDate: 'EndDate' }; |

| To map duration from data source | **Property:** *durationMapping* <br/><br/> <ej-gantt durationMapping="duration"><br/></ej-gantt> | **Property:** *taskFields.duration* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { duration: 'Duration' }; |

| To map duration unit from data source | **Property:** *durationUnitMapping* <br/><br/> <ej-gantt durationUnitMapping: "durationUnit"><br/></ej-gantt> | **Property:** *taskFields.durationUnit* <br/><br/> <ejs-gantt [taskFields]="taskSettings"><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { durationUnit: 'durationUnit' }; |



| To map predecessors from data source | **Property:** *predecessorMapping* <br/><br/> <ej-gantt predecessorMapping : “predecessor”><br/></ej-gantt> | **Property:** *taskFields.dependency* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { dependency: ‘predecessor’ } ;|

| To map progress from data source | **Property:** *progressMapping* <br/><br/> <ej-gantt progressMapping: “progress”><br/></ej-gantt> | **Property:** *taskFields.progress* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { progress: ‘progress’ } ;|

| To map child task from data source | **Property:** *childMapping* <br/><br/> <ej-gantt childMapping : “subTasks”><br/></ej-gantt> | **Property:** *taskFields.child* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { child: ‘subTasks’ } ;|

| To map baseline start date from data source | **Property:** *baselineStartDateMapping* <br/><br/> <ej-gantt baselineStartDateMapping: “baselineStartDate”><br/></ej-gantt> | **Property:** *taskFields.baselineStartDate* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { baselineStartDate: ‘baselineStartDate’ } ;|

| To map baseline end date from data source | **Property:** *baselineEndDateMapping* <br/><br/> <ej-gantt baselineEndDateMapping: “baselineEndDate”><br/></ej-gantt> | **Property:** *taskFields.baselineEndDate* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { baselineEndDate: ‘baselineEndDate’ } ;|

| To map milestone mapping from data source | **Property:** *milestoneMapping* <br/><br/> <ej-gantt milestoneMapping: “isMilestone”><br/></ej-gantt> | **Property:** *taskFields.milestone* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { milestone: ‘isMilestone’ } ;|

| To map notes from data source | **Property:** *notesMapping* <br/><br/> <ej-gantt notesMapping: “notes”><br/></ej-gantt> | **Property:** *taskFields.notes* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { notes: ‘notes’ } ;|

| To map parent task id from data source | **Property:** *parentTaskIdMapping* <br/><br/> <ej-gantt parentTaskIdMapping: “parentId”><br/></ej-gantt> | **Property:** *taskFields.parentID* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { parentId: ‘parentId’ } ;|

| To map assigned resources from data source | **Property:** *resourceInfoMapping* <br/><br/> <ej-gantt resourceInfoMapping: “assignedResource”><br/></ej-gantt> | **Property:** *taskFields.resourceInfo* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { resourceInfo: ‘assignedResource’ } ;|

| To map expand state from data source | **Property:** *expandStateMapping* <br/><br/> <ej-gantt expandStateMapping: “isExpanded”><br/></ej-gantt> | **Property:** *taskFields.expandState* <br/><br/><ejs-gantt [taskFields]=“taskSettings”><br/></ejs-gantt> <br> **TS** <br>this.taskSettings = { expandState: ‘isExpanded’ } ;|

## Sorting

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowSorting* <br/><br/> <ej-gantt [allowSorting]="true"><br/></ej-gantt> |  
**Property:** *allowSorting* <br/><br/> <ejs-gantt [allowSorting]="true"><br/></ejs-gantt> |

| To enable/disable multiple sorting option | **Property:** *allowMultiSorting* <br/><br/> <ej-gantt [allowMultiSorting]="true"><br/></ej-gantt> | **Property:** *allowSorting* <br/><br/> <ejs-gantt [allowSorting]="true"><br/></ejs-gantt> |

| Sort column Initially | **Property:** *sortSettings.sortedColumns* <br/><br/> <ej-gantt [allowSorting]="true" [sortSettings]="sortSettings"><br/></ej-gantt> <br> **TS** <br>this.sortSettings = {<br>&#160;&#160;&#160;sortedColumns : {field: "taskName", direction: "descending" } }<br>}; |  
**Property:** *sortSettings.columns* <br/><br/> <ejs-gantt [allowSorting]="true" [sortSettings]="sortSettings"><br/></ejs-gantt> <br> **TS** <br>this.sortSettings = {<br>&#160;&#160;&#160;columns: [{ field: 'TaskID', direction: 'Ascending' }]<br>}; |

| Clear the Sorted columns | **Method:** *clearSorting()* <br><br> export class AppComponent {<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public clearSorting(event) {<br>&#160;&#160;&#160;var ganttObj = \$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;&#160;ganttObj.clearSorting();<br>}<br>} | **Method:** *clearSorting()* <br><br> @ViewChild( 'gantt' )<br>public ganttObj: GanttComponent; <br>this.ganttObj.clearSorting(); |

| Sort records in Gantt | **Method:** *sortColumn(mappingName, columnSortDirection)* <br/><br/> export class AppComponent {<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public clearSorting(event) {<br>&#160;&#160;&#160;var ganttObj = \$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;&#160;ganttObj.sortColumn( "startDate", "ascending" );<br>}<br>} | **Method:** *sortColumn(columnName, direction,[isMultiSort])* <br/><br/> @ViewChild( 'gantt' )<br>public ganttObj: GanttComponent; <br>this.ganttObj.sortColumn( 'startDate', 'ascending' );<br> |

## Filtering

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Filter column Initially | **Property:** *filterSettings.filteredColumns* <br/><br/> <ej-gantt [filterSettings]="filterSettings"><br/></ej-gantt> <br> **TS** <br>this.filterSettings = {<br>&#160;&#160;&#160;filteredColumns: [ { value: "plan", field: "taskName", predicate: "and", operator: "startswith" } ]<br>}; | **Property:** *filterSettings.columns* <br/><br/> <ejs-gantt [filterSettings]="filterSettings"><br/></ejs-gantt> <br> **TS** <br>this.filterSettings = {<br>&#160;&#160;&#160;columns: [ { field: 'TaskName', matchCase: false, operator: 'startswith', predicate: 'and', value: 'Identify' } ]<br>}; |

| Filter records in Gantt | **Method:** *filterColumn(fieldName, filterOperator, filterValue, [predicate], [matchCase])* <br/><br/> export class AppComponent {<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public filterColumn(event) {<br>&#160;&#160;&#160;var ganttObj = \$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;&#160;ganttObj.filterColumn( "taskName", "startswith", "plan" );<br>}<br>} | **Method:** *filterByColumn(fieldName, filterOperator, filterValue,*

```
[predicate], [matchCase],[ignoreAccent]) <br><br>@ViewChild('gantt')<br>public ganttObj:
GanttComponent;<br>this.ganttObj.filterByColumn('taskName', 'startswith', 'plan'); |
```

```
| Filter multiple columns | Method: filterContent(ejPredicate) <br><br> export class AppComponent  
<br>constructor() {<br>#160;#160;//...<br>}<br>public filterContent(event)  
<br>#160;#160;var ganttObj = $(" "#Gantt").ejGantt("instance");<br>#160;#160;var  
predicate = ej.Predicate("taskName", ej.FilterOperators.equal, "planning",  
false)<br>#160;#160;#160;#160;.or("taskName", ej.FilterOperators.equal, "plan budget",  
false)<br>#160;#160;#160;#160;.and("progress", ej.FilterOperators.equal, 100,  
true);<br>ganttObj.filterContent(ejPredicate);<br>}<br>| Not applicable |
```

```
| Clear filtered columns | Method: clearFilter() <br/><br/>export class AppComponent  
{<br/>constructor() {<br/>&#160;&#160;&#160;...<br/>}<br/>public clearFilter(event) {<br/>&#160;&#160;&#160;var  
ganttObj = $('#Gantt').ejGantt("instance");<br/>&#160;&#160;&#160;ganttObj.clearFilter();<br/>}<br/>} |  
Method: clearFiltering() <br/><br/>@ViewChild('gantt')<br/>public ganttObj:  
GanttComponent;<br/>this.ganttObj.clearFiltering(); |
```

## Searching

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

\_\_\_\_\_

```
| Default | Property: toolbarSettings.toolbarItems <br><br> <ej-gantt
[toolbarSettings]="toolbarSettings"><br></ej-gantt> <br> TS <br>this.toolbarSettings =
{<br>&#160;&#160;showToolBar: true,<br>&#160;&#160;toolbarItems :
[ej.Gantt.ToolbarItems.Search]<br>}; | Property: toolbar <br><br> <ejs-gantt
[toolbar]="toolbar"><br></ejs-gantt> <br> TS <br>this.toolbar = ['Search'];<br>|
```

```
| Search records in Gantt | Method: searchItem(key) <br/><br/>export class AppComponent  
{<br>constructor() {<br>#160;#160;//...<br>}<br>public searchItem(event)  
{<br>#160;#160;var ganttObj =  
$(("#Gantt").ejGantt("instance");<br>#160;#160;ganttObj.searchItem("plan");<br>}<br>} |  
Method: search(key) <br/><br/>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>this.ganttObj.search('plan'); |
```

## Selection

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

\_\_\_\_\_

| Default | **Property:** `allowSelection` <br><br> <ej-gantt [allowSelection]="true"><br></ej-gantt> |  
**Property:** `allowSelection` <br><br> <ejs-gantt [allowSelection]="true"><br></ejs-gantt> |

```
| To define selection type in Gantt | Property: selectionType <br/><br/><ej-gantt
selectionType="single"<br/></ej-gantt>| Property: selectionSettings.type <br/><br/><ejs-gantt
[selectionSettings]="selectionSettings"><br/></ejs-gantt> <br/> TS <br/>this.selectionSettings = {
type: 'Multiple' };
```

| To define selection mode in Gantt | **Property:** *selectionMode* <br/><br/> <ej-gantt  
selectionMode="cell"<br/></ej-gantt> | **Property:** *selectionSettings.mode* <br/><br/> <eis-gantt

```
[selectionSettings]="selectionSettings"></ej-gantt> <br> TS <br>this.selectionSettings = {
mode: 'Cell' };
```

| Select Row by Index | **Property:** *selectedRowIndex* <br><br> <ej-gantt>

```
[selectedRowIndex]="selectedRowIndex"></ej-gantt> <br> TS <br>this.selectedRowIndex =
3 | Property: selectedRowIndex <br><br><ej-gantt [selectedRowIndex]='5'></ej-gantt> |
```

| To define selected cell index in Gantt | **Property:** *selectedCellIndexes* <br><br> <ej-gantt>

```
[selectedCellIndexes]="selectedCellIndexes"></ej-gantt><br> TS
<br>this.selectedCellIndexes = [ ]; | Not applicable |
```

| Select Multiple Cells | **Method:** *selectCells(Indexes,preservePreviousSelectedCell)* <br><br> export  
class AppComponent {<br>constructor() {<br>...<br>}<br>public selectCells(event)  
{<br>...<br>var ganttObj = \$("#Gantt").ejGantt("instance");<br>...<br>var indexes  
= [{rowIndex:4, cellIndex: 4}, {rowIndex: 3, cellIndex: 3}];<br>ganttObj.selectCells(indexes,  
true);| Not Applicable |

| Select multiple Rows | **Method:** *selectMultipleRows(rowIndexes)* <br><br>export class  
AppComponent {<br>constructor() {<br>...<br>}<br>public  
selectMultipleRows(event) {<br>...<br>var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>...<br>ganttObj.selectMultipleRows([1,2,3]);<br>}<br>| **Method:** *selectRows(key)* <br><br>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>this.ganttObj.selectionModule.selectRows([1,2,3]); |

| Triggers after cell selection action | **Event:** *cellSelected* <br><br><ej-gantt id="Gantt"  
(cellSelected)="cellSelected(\$event)"></ej-gantt><br> TS <br>cellSelected(event) { } | **Event:**  
*cellSelected* <br><br><ejs-gantt (cellSelected)="cellSelected(\$event)"></ejs-gantt><br> TS  
<br>public cellSelected(event) { } |

| Triggers on cell selection action | **Event:** *cellSelecting* <br><br><ej-gantt id="Gantt"  
(cellSelecting)="cellSelecting(\$event)"></ej-gantt><br> TS <br>cellSelecting(event) { } |  
**Event:** *cellSelecting* <br><br><ejs-gantt (cellSelecting)="cellSelecting(\$event)"></ejs-  
gantt><br> TS <br>public cellSelecting(event) { } |

| Triggers after row selection action | **Event:** *rowSelected* <br><br><ej-gantt id="Gantt"  
(rowSelected)="rowSelected(\$event)"></ej-gantt><br> TS <br>rowSelected(event) { } |  
**Event:** *rowSelected* <br><br><ejs-gantt (rowSelected)="rowSelected(\$event)"></ejs-  
gantt><br> TS <br>public rowSelected(event) { } |

| Triggers before row selection action | **Event:** *rowSelecting* <br><br><ej-gantt id="Gantt"  
(rowSelecting)="rowSelecting(\$event)"></ej-gantt><br> TS <br>rowSelecting(event) { } |  
**Event:** *rowSelecting* <br><br><ejs-gantt (rowSelecting)="rowSelecting(\$event)"></ejs-  
gantt><br> TS <br>public rowSelecting(event) { } |

## Editing

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *editSettings* <br><br> <ej-gantt [editSettings]= "editSettings"><br></ej-gantt>  
 <br> **TS** <br>this.editSettings = {<br>&#160;&#160;allowEditing:  
 true,<br>&#160;&#160;allowAdding: true,<br>&#160;&#160;allowDeleting:  
 true,<br>&#160;&#160;showDeleteConfirmDialog: true};<br>}; | **Property:** *editSettings* <br><br>  
 <ejs-gantt [editSettings]= "editSettings"><br></ejs-gantt><br> **TS** <br>this.editSettings =  
 {<br>&#160;&#160;allowEditing: true,<br>&#160;&#160;allowAdding:  
 true,<br>&#160;&#160;allowDeleting: true,<br>&#160;&#160;showDeleteConfirmDialog: true};|

| Cell Editing | **Property:** *editSettings.editMode* <br><br> <ej-gantt  
 [editSettings]= "editSettings"><br></ej-gantt> <br> **TS** <br>this.editSettings = { editMode:  
 "cellEditing" }; | **Property:** *editSettings.mode* <br><br> <ejs-gantt  
 [editSettings]= "editSettings"><br></ejs-gantt> <br> **TS** <br>this.editSettings = { mode: "Auto" };|

| Dialog Editing | **Property:** *editSettings.editMode* <br><br> <ej-gantt  
 [editSettings]= "editSettings"><br></ej-gantt> <br> **TS** <br>this.editSettings = { editMode:  
 "normal" }; | **Property:** *editSettings.mode* <br><br> <ejs-gantt  
 [editSettings]= "editSettings"><br></ejs-gantt> <br> **TS** <br>this.editSettings = { mode: "Dialog"  
 };|

| To enable/disable taskbar editing | **Property:** *allowGanttChartEditing* <br><br> <ej-gantt  
 [allowGanttChartEditing]= "true"><br></ej-gantt> | **Property:** *editSettings.allowTaskbarEditing*  
 <br><ejs-gantt [editSettings]= "editSettings"><br></ejs-gantt> <br> **TS** <br>this.editSettings = {  
 allowTaskbarEditing: true };|

| To enable progressbar resizing | **Property:** *enableProgressBarResizing* <br><br> <ej-gantt  
 [enableProgressBarResizing]= "true"><br></ej-gantt> | **Property:** *editSettings.allowTaskbarEditing*  
 <br><br> <ejs-gantt [editSettings]= "editSettings"><br></ejs-gantt> <br> **TS** <br>this.editSettings  
 = { allowTaskbarEditing: true };|

| To enable indent/ outdent option | **Property:** *editSettings.allowIndent* <br><br> <ej-gantt  
 [editSettings]= "editSettings"><br></ej-gantt> <br> **TS** <br>this.editSettings = { allowIndent: true  
 };| Not applicable |

| To define click or double click action to begin edit action | **Property:** *editSettings.beginEditAction* <br><br>  
 <br> <ej-gantt [editSettings]= "editSettings"><br></ej-gantt> <br> **TS** <br>this.editSettings = {  
 beginEditAction: 'click' };| Not applicable |

| To define new row position in Gantt | **Property:** *editSettings.rowPosition* <br><br> <ej-gantt  
 [editSettings]= "editSettings"><br></ej-gantt> <br> **TS** <br>this.editSettings = { rowPosition:  
 ej.Gantt.RowPosition.AboveSelectedRow}; | **Property:** *editSettings.newRowPosition* <br><br> <ejs-  
 gantt [editSettings]= "editSettings"><br></ejs-gantt> <br> **TS** <br>this.editSettings = {  
 newRowPosition: 'Below'};|

| To define fields in edit dialog | **Property:** *editDialogFields* <br><br> <ej-gantt [editDialogFields]=  
 "editDialogFields"><br></ej-gantt> <br> **TS** <br>this.editDialogFields = [{ field: 'taskName',  
 editType: 'stringedit' }]; | **Property:** *editDialogFields* <br><br> <ejs-gantt  
 [editDialogFields]= "editDialogFields"><br></ejs-gantt> <br> **TS** <br>this.editDialogFields = [ {  
 type: 'General', fields: ['TaskName'] } ];|

| To define fields in add dialog | **Property:** *addDialogFields* <br/> <br/> <ej-gantt  
[addDialogFields]="addDialogFields"><br/></ej-gantt><br> **TS** <br>this.addDialogFields = [{ field:  
'taskName', editType: 'stringedit' }]; | **Property:** *addDialogFields* <br/><br/> <ejs-gantt  
[addDialogFields]="addDialogFields"><br/></ejs-gantt><br> **TS** <br>this.addDialogFields: [ { type:  
'General', fields: ['taskName'] } ]; |

| To make Gantt as read only | **Property:** *readOnly* <br/> <br/> <ej-gantt [readOnly]=  
"true"><br/></ej-gantt> | Not Applicable |

| To open Edit dialog | **Method:** *openEditDialog()* <br/><br/>export class AppComponent  
{<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public openEditDialog(event)  
{<br>&#160;&#160;&#160;var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>&#160;&#160;&#160;ganttObj.openEditDialog();<br>}<br>} | **Method:** *openEditDialog()* <br/><br/>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>this.ganttObj.openEditDialog(); |

| To open Add dialog | **Method:** *openAddDialog()* <br/><br/>export class AppComponent  
{<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public openAddDialog(event)  
{<br>&#160;&#160;&#160;var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>&#160;&#160;&#160;ganttObj.openAddDialog();<br>}<br>} | **Method:** *openAddDialog()* <br/><br/>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>this.ganttObj.openAddDialog(); |

| Add task in Gantt | **Method:** *addRecord(data, rowPosition)* <br/><br/>export class AppComponent  
{<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public addRecord(event) {<br>var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>var data =  
{<br>&#160;&#160;&#160;taskId:"40",<br>&#160;&#160;&#160;taskName:"New Task  
40",<br>&#160;&#160;&#160;startDate:"2/20/2014",<br>&#160;&#160;&#160;endDate:"2/25/2014"<br>}<br>ganttObj.addRecord(data, ej.Gantt.AddRowPosition.Child);<br>}<br>} | **Method:**  
*addRecord(data, rowPosition, rowIndex)* <br/><br/>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>var data =  
{<br>&#160;&#160;&#160;taskId:"40",<br>&#160;&#160;&#160;taskName:"New Task  
40",<br>&#160;&#160;&#160;startDate:"2/20/2014",<br>&#160;&#160;&#160;endDate:"2/25/2014"<br>}<br>this.ganttObj.addRecord(data, 'Below', 10); |

| Delete selected item | **Method:** *deleteItem()* <br/><br/>export class AppComponent  
{<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public deleteItem(event)  
{<br>&#160;&#160;&#160;var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>&#160;&#160;&#160;ganttObj.deleteItem();<br>}<br>} | **Method:**  
*deleteRecord()* <br/><br/>@ViewChild('gantt')<br>public ganttObj:  
GanttComponent;<br>this.ganttObj.editModule.deleteRecord(); |

| Update task details by id | **Method:** *updateRecordByTaskId(data)* <br/><br/>export class  
AppComponent {<br>constructor() {<br>&#160;&#160;&#160;...<br>}<br>public  
updateRecordByTaskId(event) {<br>&#160;&#160;&#160;var ganttObj =  
\$("#Gantt").ejGantt("instance");<br>&#160;&#160;&#160;var data = { taskId: 4, taskName: "updated  
value"};<br>ganttObj.updateRecordByTaskId(data);<br>}<br>} | **Method:** *updateRecordById*



```
</></>@ViewChild('gantt')<br>public ganttObj: GanttComponent;<br>var data = { taskID: 4,  
taskName: "updated value"};<br>this.ganttObj.updateRecordByID(data); |
```

```
| Delete dependency | Method: deleteDependency(fromTaskId,toTaskId) <br/><br/>export class  
AppComponent {<br>constructor() {<br>#160;#160;#160;//...<br>}<br>public  
deleteDependency(event) {<br>#160;#160;var ganttObj =  
$(("#Gantt").ejGantt("instance");<br>#160;#160;ganttObj.deleteDependency(3, 6); | Not  
applicable |
```

Save Edit	Method: <code>saveEdit()</code>	<pre> export class AppComponent {   constructor() {     //...   }   public saveEdit(event) {     var ganttObj = \$( "#Gantt" ).ejGantt( "instance" );     ganttObj.saveEdit();   } } </pre>	Not applicable
-----------	---------------------------------	---	----------------

```
| Cancel Edit | Method: cancelEdit() <br><br> export class AppComponent {<br><br> constructor()
{<br>&#160;&#160;&#160;...<br><br> public cancelEdit(event) {<br>&#160;&#160;&#160;var ganttObj =
$(“#Gantt”).ejGantt(“instance”);<br>&#160;&#160;&#160;ganttObj.cancelEdit(); | Method: cancelEdit()
<br><br> @ViewChild(‘gantt’)<br> public ganttObj: GanttComponent;
<br> this.ganttObj.cancelEdit()
```

```
|Triggers for every Gantt action before it get started | Event: actionBegin <br/><br/><ej-gantt
id="Gantt" (actionBegin)="actionBegin($event)"><br/></ej-gantt><br> TS <br>actionBegin(event)
{ } | Event: actionBegin <br/><br/><ejs-gantt (actionBegin)="actionBegin($event)"><br/></ejs-
gantt><br> TS <br>public actionBegin(event) { } |
```

```
|Triggers for after every Gantt action completed | Event: actionComplete <br/><br/><ej-gantt
id="Gantt" (actionComplete)="actionComplete($event)"><br/></ej-gantt><br> TS
<br>actionComplete(event) { } | Event: actionComplete <br/><br/><ejs-gantt
(actionComplete)="actionComplete($event)"><br/></ejs-gantt><br> TS <br>public
actionComplete(event) { } |
```

```
| Triggers while resizing, dragging the taskbar | Event: taskbarEditing <br/><br/><ej-gantt id="Gantt"
(taskbarEditing)="taskbarEditing($event)"><br/></ej-gantt><br/> TS <br/>taskbarEditing(event) { } |
Event: taskbarEditing <br/><br/><ejs-gantt (taskbarEditing)="taskbarEditing($event)"><br/></ejs-
gantt><br/> TS <br/>public taskbarEditing(event) { } |
```

```
| Triggers after taskbar resize, drag action | Event: taskbarEdited <br/><br/><ej-gantt id="Gantt"
(taskbarEdited)="taskbarEdited($event)"><br/></ej-gantt><br/> TS <br/>taskbarEdited(event) { } |
Event: taskbarEdited <br/><br/><ejs-gantt (taskbarEdited)="taskbarEdited($event)"><br/></ejs-
gantt><br/> TS <br/>public taskbarEdited(event) { } |
```

## Columns

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

\_\_\_\_\_

| To enable/disable column resize | **Property:** `allowColumnResize` <br/> <br/> <ej-gantt  
[allowColumnResize]="true"><br/></ej-gantt> | **Property:** `allowResizing` <br/><br/> <ejs-gantt  
[allowResizing]="true"><br/><ejs/gantt> |

| To enable/disable column chooser | **Property:** `showColumnChooser` <br/><br/> <ej-gantt  
[showColumnChooser]="true"><br/></ej-gantt> | **Property:** `showColumnMenu` <br/><br/> <ejs-gantt  
[showColumnMenu]="true"><br/><ejs/gantt> |

<p>  To enable/disable column add, remove option in column menu  <b>Property:</b> <i>showColumnOptions</i></p> <p>&lt;br/&gt;&lt;br/&gt; &lt;ej-gantt [showColumnOptions]="true"&gt;&lt;br/&gt;&lt;/ej-gantt&gt;   Not applicable  </p>
---

| Tree column index | **Property:** *treeColumnIndex* <br/> <br/> <ej-gantt [treeColumnIndex]=  
"1"><br></ej-gantt> | **Property:** *treeColumnIndex* <br/> <br/> <ejs-gantt [treeColumnIndex]=  
"2"><br></ejs-gantt> |

<p>  To define column fields in column menu   <b>Property:</b> <code>columnDialogFields</code> &lt;br/&gt;&lt;br/&gt;&lt;ej-gantt [columnDialogFields]=“columnDialogFields”&gt;&lt;br/&gt;&lt;/ej-gantt&gt;&lt;br/&gt; <b>TS</b> &lt;br/&gt;this.columnDialogFields = [“field”, “headerText”, “editType”, “width”, “visible”, “allowSorting”, “textAlign”, “headerTextAlign”];   Not applicable  </p>	
---	--

```
| Show column | Method: showColumn(headerText) <br><br> export class AppComponent
{<br>constructor() {<br>&#160;&#160;{//...<br>}<br>public showColumn(event)
{<br>&#160;&#160;var ganttObj =
$("#Gantt").ejGantt("instance");<br>&#160;&#160;ganttObj.showColumn("Task Name"); |
Method: showColumn(keys, showBy) <br><br> @ViewChild('gantt')<br>public ganttObj:
GanttComponent; <br>this.ganttObj.showColumn("TaskName"); |
```

```
| Hide column | Method: hideColumn(headerText) <br><br> export class AppComponent
{<br>constructor() {<br>&#160;&#160;///...<br>}<br>public hideColumn(event)
{<br>&#160;&#160;var ganttObj =
$("#Gantt").ejGantt("instance");<br>&#160;&#160;ganttObj.hideColumn("Task Name"); |
Method: hideColumn(keys, showBy) <br><br> @ViewChild('gantt')<br>public ganttObj:
GanttComponent; <br>this.ganttObj.hideColumn("TaskName"); |
```

## Toolbar

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

\_\_\_\_\_

```
| To configure default toolbars of Gantt | Property: toolbarSettings.toolbarItems <br/> <br/> <ej-gantt
[toolbarSettings]= "toolbarSettings"><br></ej-gantt><br> TS <br>this.toolbarSettings =
{<br>&#160;&#160;showToolBar: true,<br>&#160;&#160;toolbarItems:
[<br>&#160;&#160;ej.Gantt.ToolbarItems.Add,<br>&#160;&#160;ej.Gantt.ToolbarItems.Edit,<br>
&#160;&#160;ej.Gantt.ToolbarItems.Delete,<br>&#160;&#160;ej.Gantt.ToolbarItems.Update,<br>
&#160;&#160;ej.Gantt.ToolbarItems.Cancel,<br>&#160;&#160;ej.Gantt.ToolbarItems.ExpandAll,<br>
&#160;&#160;ej.Gantt.ToolbarItems.CollapseAll,<br>&#160;&#160;ej.Gantt.ToolbarItems.Sear
ch,<br>&#160;&#160;ej.Gantt.ToolbarItems.PrevTimeSpan,<br>&#160;&#160;ej.Gantt.ToolbarIte
ms.NextTimeSpan<br>],<br>]<br></ej-gantt> | Property: toolbar <br/> <br/> <ejs-gantt
[toolbar]= "toolbar"><br></ejs-gantt><br> TS <br>this.toolbar =
['Add','Edit','Delete','Update','Cancel','ExpandAll',<br>&#160;&#160;'CollapseAll','Search','PrevT
imeSpan','NextTimeSpan']; |
```



| Other toolbars | **Property:** *toolbarSettings.toolbarItems* <br/> <br/> <ej-gantt [toolbarSettings]=  
 “toolbarSettings”><br/></ej-gantt><br> **TS** <br>this.toolbarSettings =  
 {<br>&#160;&#160;showToolbar: true,<br>&#160;&#160;toolbarItems:  
 [<br>&#160;&#160;ej.Gantt.ToolbarItems.Indent,<br>&#160;&#160;ej.Gantt.ToolbarItems.Outdent,<br>&#160;&#160;ej.Gantt.ToolbarItems.CriticalPath,<br>&#160;&#160;ej.Gantt.ToolbarItems.ExcelExport,<br>&#160;&#160;ej.Gantt.ToolbarItems.PdfExport<br>];| Not applicable |

| Custom toolbar | **Property:** *toolbarSettings.customToolbarItems* <br/> <br/> <ej-gantt  
 [toolbarSettings]= “toolbarSettings”><br/></ej-gantt><br> **TS** <br>this.toolbarSettings =  
 {<br>&#160;&#160;showToolbar: true,<br>&#160;&#160;customToolbarItems: [{ text:  
 “ShowBaseline”, tooltipText: “Show Baseline” }, { text: “Reset”, tooltipText: “Reset” } ]<br>}; |  
**Property:** *toolbar* <br/><br/> <ejs-gantt [toolbar]=“toolbar”><br/></ejs-gantt><br> **TS**  
 <br>this.toolbar = [{text: ‘Quick Filter’, tooltipText: ‘Quick Filter’, id: ‘toolbarfilter’,  
 align:‘Right’}]; |

| Triggers when toolbar items clicked | **Event:** *toolbarClick* <br/><br/><ej-gantt id=“Gantt”  
 (toolbarClick)=“toolbarClick(\$event)”><br/></ej-gantt><br> **TS** <br>toolbarClick(event) { } | **Event:**  
*toolbarClick* <br/><br/><ejs-gantt (toolbarClick)=“toolbarClick(\$event)”><br/></ejs-gantt><br> **TS**  
 <br>public toolbarClick(event) { } |

## ToolTip

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To enable taskbar tooltip | **Property:** *enableTaskbarTooltip* <br/> <br/> <ej-gantt  
 [enableTaskbarTooltip]=“true”><br/></ej-gantt>| **Property:** *tooltipSettings.showTooltip* <br/><br/>  
 <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br> **TS** <br>this.tooltipSettings =  
 { showTooltip: true } > |

| To define tooltip for all cells | **Property:** *cellTooltipTemplate* <br/><br/> <ej-gantt  
 cellTooltipTemplate=“#CustomToolTip”><br/></ej-gantt>| Not applicable |

| To define tooltip template for row drag action | **Property:** *dragTooltip* <br/><br/> <ej-gantt  
 [dragTooltip]= “dragTooltip”><br/></ej-gantt><br> **TS** <br>this.dragTooltip = { showTooltip: true  
 };| Not applicable |

| To enable taskbar editing tooltip | **Property:** *enableTaskbarDragTooltip* <br/> <br/> <ej-gantt  
 [enableTaskbarDragTooltip]=“true”><br/></ej-gantt>| Not Applicable |

| To enable/disable tooltip for grid cells | **Property:** *showGridCellTooltip* <br/><br/> <ej-gantt  
 [showGridCellTooltip]=“true”><br/></ej-gantt>| Not applicable |

| To enable/disable tooltip for grid cells | **Property:** *showGridExpandCellTooltip* <br/><br/> <ej-gantt  
 [showGridExpandCellTooltip]=“true”><br/></ej-gantt>| Not applicable |

| To define taskbar tooltip template in Gantt | **Property:** *taskbarTooltipTemplate* <br/> <br/> <ej-gantt  
 taskbarTooltipTemplate=“template tooltip string”><br/></ej-gantt>| **Property:**  
*tooltipSettings.taskbar* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-  
 gantt><br> **TS** <br>this.tooltipSettings = { taskbar: “template tooltip string:” } > |

| To define taskbar tooltip template id in Gantt | **Property:** *taskbarTooltipTemplateId* <br/> <br/> <ej-gantt taskbarTooltipTemplateId = “#templateId”><br/></ej-gantt>| **Property:** *tooltipSettings.taskbar* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { taskbar : “#templateId” } > |

| To define tooltip template for connector line | **Property:** *predecessorTooltipTemplate* <br/> <br/> <ej-gantt predecessorTooltipTemplate= “predecessorTooltipTemplate”><br/></ej-gantt>| **Property:** *tooltipSettings.connectorLine* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { connectorLine : “predecessorTooltip” }; |

| To define template for progress resize tooltip | **Property:** *progressbarTooltipTemplate* <br/> <br/> <ej-gantt progressbarTooltipTemplate= “progressbarTooltipTemplate”><br/></ej-gantt>| **Property:** *tooltipSettings.editing* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { editing : “template tooltip string” }; |

| To define template id for progress resize tooltip | **Property:** *progressbarTooltipTemplateId* <br/> <br/> <ej-gantt progressbarTooltipTemplateId= “#tooltipTemplateID”><br/></ej-gantt>| **Property:** *tooltipSettings.editing* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { editing : “#progressResize” }; |

| To define tooltip template for taskbar edit action | **Property:** *taskbarEditingTooltipTemplate* <br/> <br/> <ej-gantt taskbarEditingTooltipTemplate = “tooltipTemplate”><br/></ej-gantt>| **Property:** *tooltipSettings.editing* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { editing : “template tooltip string” }; |

| To define tooltip template id for taskbar edit action | **Property:** *taskbarEditingTooltipTemplateId* <br/> <br/> <ej-gantt taskbarEditingTooltipTemplateId = “#templateId”><br/></ej-gantt>| **Property:** *tooltipSettings.editing* <br/><br/> <ejs-gantt [tooltipSettings]=“tooltipSettings”><br/></ejs-gantt><br/> **TS** <br/>this.tooltipSettings = { editing : “#templateId” } > |

## Timeline

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To configure timeline settings in Gantt | **Property:** *scheduleHeaderSettings* <br/><br/> <ej-gantt [scheduleHeaderSettings]=“scheduleHeaderSettings”><br/></ej-gantt><br/> **TS** <br/>this.scheduleHeaderSettings =<br/>{<br/>weekHeaderFormat: “MMM dd , yyyy”,<br/>dayHeaderFormat: “dd,MM,yy”, <br/>yearHeaderFormat: “yyyy”,<br/>monthHeaderFormat: “MMM”,<br/>hourHeaderFormat: “HH”,<br/>scheduleHeaderType: “week”, <br/>minutesPerInterval: “auto”,<br/>weekendBackground: “#F2F2F2”,<br/>timescaleStartDateMode: “auto”, <br/>timescaleUnitSize: “100%”, <br/>weekStartDay: 0,<br/>updateTimescaleView: true <br/>}<br/></ej-gantt>| **Property:** *timelineSettings* <br/><br/> <ejs-gantt [timelineSettings]=“timelineSettings”><br/></ejs-gantt><br/> **TS** <br/>this.timelineSettings = {<br/>#160;timelineViewMode: ‘Week’,<br/>#160;timelineUnitSize: 33,<br/>#160;weekStartDay: 0,<br/>#160;showTooltip: true,<br/>#160;weekendBackground: ‘ ‘<br/>#160;updateTimescaleView: true,<br/>#160;topTier: {<br/>#160;#160;#160;unit: ‘Week’,<br/>#160;#160;#160;format: ‘MMM dd , y’,<br/>#160;#160;#160;count: 1,<br/>#160;#160;#160;formatter: null<br/>},<br/>#160;bottomTier:

```
{<br/>&#160;&#160;&#160;unit: 'Day',<br/>&#160;&#160;&#160;format:
'dd',<br/>&#160;&#160;&#160;count: 1,<br/>&#160;&#160;&#160;formatter: null<br/>}<br/>;|

| To define weekend background in Gantt | Property: weekendBackground <br/> <br/> <ej-gantt
weekendBackground= "blue"><br/></ej-gantt>; | Not applicable|

| To Highlight weekends | Property: highlightWeekends <br/><br/> <ej-gantt
[highlightWeekends]="true"><br/></ej-gantt>; | Property: highlightWeekends <br/><br/> <ejs-gantt
[highlightWeekends]="true"><br/></ejs-gantt> |

| To include weekends | Property: includeWeekend <br/><br/> <ej-gantt
[includeWeekend]="true"><br/></ej-gantt>; | Property: includeWeekend <br/><br/> <ejs-gantt
[includeWeekend]="true"><br/></ejs-gantt> |

| To define project start date in Gantt | Property: scheduleStartDate <br/><br/> <ej-gantt
scheduleStartDate = "3/20/2018"><br/></ej-gantt>| Property: projectStartDate <br/><br/> <ejs-
gantt [projectStartDate]="projectStartDate"><br/></ej-gantt><br/> TS <br/>this. projectStartDate =
new Date('3/20/2018');|

| To define project end date in Gantt | Property: scheduleEndDate <br/><br/> <ej-gantt
scheduleEndDate = "3/20/2018"><br/></ej-gantt>| Property: projectEndDate <br/><br/> <ejs-gantt
[projectEndDate]="projectEndDate"><br/></ej-gantt><br/> TS <br/>this. projectEndDate = new
Date('3/20/2018');|

| Update project start date and end date | Method: updateScheduleDates(startDate,endDate)
<br/><br/>export class AppComponent {<br/>constructor() {<br/>&#160;&#160;&#160;...<br/>}<br/>public
updateScheduleDates(event) {<br/>&#160;&#160;&#160;var ganttObj =
$( "#Gantt" ).ejGantt( "instance" );<br/>&#160;&#160;&#160;ganttObj.updateScheduleDates( "5/25/2017",
"9/27/2017" );<br/>}<br/>| Method: updateProjectDates(startDate, endDate, isTimelineRoundOff)
<br/><br/>@ViewChild('gantt')<br/>public ganttObj:
GanttComponent;<br/>this.ganttObj.updateProjectDates( "5/25/2017", "9/27/2017", true ); |
```

## Rows

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

```
| To enable/disable row drag and drop | Property: allowDragAndDrop <br/> <br/> <ej-gantt
[allowDragAndDrop]="true"><br/></ej-gantt> | Not applicable |

| To enable/disable alternate row background | Property: enableAltRow <br/> <br/> <ej-gantt
[enableAltRow]="true"><br/></ej-gantt> | Not applicable |

| To add Row height | Property: rowHeight <br/><br/> <ej-gantt [rowHeight]="60"><br/></ej-gantt>|
Property: rowHeight <br/><br/> <ejs-gantt [rowHeight]="60"><br/></ejs-gantt> |

| To render parent in collapsed state | Property: enableCollapseAll <br/> <br/> <ej-gantt
[enableCollapseAll]="true"><br/></ej-gantt>| Property: collapseAllParentTasks <br/><br/> <ejs-
gantt [collapseAllParentTasks]="true" > |

| Expand/collapse record by id | Method: expandCollapseRecord(taskId) <br/><br/>export class
AppComponent {<br/>constructor() {<br/>&#160;&#160;&#160;...<br/>}<br/>public
```

```
expandCollapseRecord(event) {<br>&#160;&#160;var ganttObj =
$(“#Gantt”).ejGantt(“instance”);<br>&#160;&#160;ganttObj.expandCollapseRecord(1);<br>}<br>
} | Method: collapseById() expandById() <br><br>@ViewChild(‘gantt’)<br>public ganttObj:
GanttComponent;<br>this.ganttObj.expandById(1);<br><br>this.ganttObj.collapseById(1); |
```

```
| Expand all rows | Method: expandAllItems() <br><br>export class AppComponent
{<br>constructor() {<br>&#160;&#160;...<br>}<br>public expandAllItems(event)
{<br>&#160;&#160;var ganttObj =
$(“#Gantt”).ejGantt(“instance”);<br>&#160;&#160;ganttObj.expandAllItems();<br>}<br>} |
Method: expandAll() <br><br>@ViewChild(‘gantt’)<br>public ganttObj:
GanttComponent;<br>this.ganttObj.expandAll(); |
```

```
| Collapse all rows | Method: collapseAllItems() <br><br>export class AppComponent
{<br>constructor() {<br>&#160;&#160;...<br>}<br>public collapseAllItems(event)
{<br>&#160;&#160;var ganttObj =
$(“#Gantt”).ejGantt(“instance”);<br>&#160;&#160;ganttObj.collapseAllItems();<br>}<br>} |
Method: collapseAll() <br><br>@ViewChild(‘gantt’)<br>public ganttObj:
GanttComponent;<br>this.ganttObj.collapseAll(); |
```

```
| Triggers after row collapse action | Event: collapsed <br><br><ej-gantt id=“Gantt”
(collapsed)=“collapsed($event)”><br></ej-gantt><br> TS <br>collapsed(event) { } | Event:
collapsed <br><br><ejs-gantt (collapsed)=“collapsed($event)”><br></ejs-gantt><br> TS
<br>public collapsed(event) { } |
```

```
| Triggers before row collapse action | Event: collapsing <br><br><ej-gantt id=“Gantt”
(collapsing)=“collapsing($event)”><br></ej-gantt><br> TS <br>collapsing(event) { } | Event:
collapsing <br><br><ejs-gantt (collapsing)=“collapsing($event)”><br></ejs-gantt><br> TS
<br>public collapsing(event) { } |
```

```
| Triggers after Gantt row was expanded | Event: expanded <br><br><ej-gantt id=“Gantt”
(expanded)=“expanded($event)”><br></ej-gantt><br> TS <br>expanded(event) { } | Event:
expanded <br><br><ejs-gantt (expanded)=“expanded($event)”><br></ejs-gantt><br> TS
<br>public expanded(event) { } |
```

```
| Triggers before Gantt row expand action | Event: expanding <br><br><ej-gantt id=“Gantt”
(expanding)=“expanding($event)”><br></ej-gantt><br> TS <br>expanding(event) { } | Event:
expanding <br><br><ejs-gantt (expanding)=“expanding($event)”><br></ejs-gantt><br> TS
<br>public expanding(event) { } |
```

```
| Triggers before grid rows render action | Event: rowDataBound <br><br><ej-gantt id=“Gantt”
(rowDataBound)=“rowDataBound($event)”><br></ej-gantt><br> TS <br>rowDataBound(event) {
} | Event: rowDataBound <br><br><ejs-gantt
(rowDataBound)=“rowDataBound($event)”><br></ejs-gantt><br> TS <br>public
rowDataBound(event) { } |
```

```
| Triggers while dragging a row | Event: rowDrag <br><br><ej-gantt id=“Gantt”
(rowDrag)=“rowDrag($event)”><br></ej-gantt><br> TS <br>rowDrag(event) { } | Not applicable |
```

| Triggers while while start to drag row | **Event:** *rowDragStart* <br/><br/><ej-gantt id="Gantt"  
(rowDragStart)="rowDragStart(\$event)"><br/></ej-gantt><br/> **TS** <br/>rowDrag(event) { } | Not  
applicable |

| Triggers while while drop a row | **Event:** *rowDragStop* <br/><br/><ej-gantt id="Gantt"  
(rowDragStop)="rowDragStop(\$event)"><br/></ej-gantt><br/> **TS** <br/>rowDrag(event) { } | Not  
applicable |

## Resources

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To map resources | **Property:** *resources* <br/><br/><ej-gantt [resources]="resources"><br/></ej-  
gantt><br/> **TS** <br/>this.resources = [{id:1; name:"jack" }]; | **Property:** *resources* <br/><br/><ejs-gantt  
[resources]="resources"><br/></ejs-gantt><br/> **TS** <br/>this.resources = [{ resourceId: 1,  
resourceName: 'Martin Tamer' }]; |

| To map resource id field from resource collection | **Property:** *resourceIdMapping* <br/><br/><ej-  
gantt resourceIdMapping = "id"><br/></ej-gantt> | **Property:** *resourceIdMapping* <br/><br/><ejs-  
gantt resourceIdMapping='resourceId'><br/></ejs-gantt> |

| To map resource name field from resource collection | **Property:** *resourceNameMapping* <br/><br/><ej-  
gantt resourceNameMapping = "name"><br/></ej-gantt> | **Property:** *resourceNameMapping*  
<br/><br/><ejs-gantt resourceNameMapping='resourceName'><br/></ejs-gantt> |

| To map resource unit field from assigned resource collection | **Property:** *resourceUnitMapping* <br/>  
<br/><ej-gantt resourceUnitMapping = "Unit"><br/></ej-gantt> | Not applicable |

| To define resource view type of Gantt | **Property:** *viewType* <br/><br/><ej-gantt [viewType]=  
"viewType"><br/></ej-gantt><br/> **TS** <br/>this.viewType =  
ej.Gantt.ViewType.ResourceView><br/></ej-gantt> | Not applicable |

| To define mapping property for resource collection in resource view Gantt | **Property:**  
*resourceCollectionMapping* <br/><br/><ej-gantt resourceCollectionMapping=  
"resources"><br/></ej-gantt> | Not Applicable |

| To map task collection from resources for resource view Gantt | **Property:** *taskCollectionMapping*  
<br/><br/><ej-gantt taskCollectionMapping= "tasks"><br/></ej-gantt> | Not applicable |

| To map group id for resource view Gantt | **Property:** *groupIdMapping* <br/><br/><ej-gantt  
groupIdMapping= "groupId"><br/></ej-gantt> | Not Applicable |

| To map group name for resource view Gantt | **Property:** *groupNameMapping* <br/><br/><ej-gantt  
groupNameMapping= "groupName"><br/></ej-gantt> | Not Applicable |

## Baseline

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To render baseline | **Property:** `renderBaseline` <br><br> <ej-gantt [renderBaseline] = "true"><br></ej-gantt> | **Property:** `renderBaseline` <br><br> <ejs-gantt [renderBaseline]="true"><br></ejs-gantt> |

| To define baselineColor | **Property:** *baselineColor* <br><br><ej-gantt [baselineColor]=  
"blue"><br></ej-gantt>| **Property:** *baselineColor* <br><br><ejs-gantt  
baselineColor="red"><br></ejs-gantt> |

## Context Menu

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

-----

| To enable context menu | **Property:** *enableContextMenu* <br><br> <ej-gantt  
[enableContextMenu]="true"><br></ej-gantt> | **Property:** *enableContextMenu* <br><br> <ejs-gantt  
[enableContextMenu]="true" > |

```
| To define custom menu items | Event: contextMenuOpen <br><br></ej-gantt id="Gantt"
(contextMenuOpen)="contextMenuOpen($event)"><br></ej-gantt><br> TS
<br>contextMenuOpen(event)
{<br><br><br>event.contextMenuItems.push({<br><br><br>headerText:
"Expand/Collapse",<br><br><br><br>menuId:
"expand",<br><br><br><br>iconPath: "url(Expand-02-
WF.png)",<br><br><br><br>eventHandler: function()
{<br><br><br><br><br><br>eventHandler for custom menu
items<br><br><br><br><br>}};<br><br>}} | Property: contextMenuItems <br><br></ej-gantt
[contextMenuItems]="contextMenuItems"><br></ej-gantt><br> TS <br>this.contextMenuItems
= [ <br><br>{ text: 'Collapse the Row', target: '.e-content', id: 'collapserow' } as
ContextMenuItemModel,<br><br>{ text: 'Expand the Row', target: '.e-content', id:
'expandrow' } as ContextMenuItemModel<br>];|
```

```
| Triggers before context menu opens | Event: contextMenuOpen <br/><br/><ej-gantt id="Gantt"
(contextMenuOpen)="contextMenuOpen($event)"><br/></ej-gantt><br> TS
<br/>contextMenuOpen(event) { } | Event: contextMenuOpen <br/><br/><ejs-gantt
(contextMenuOpen)="contextMenuOpen($event)"><br/></ejs-gantt><br> TS <br/>public
contextMenuOpen(event) { } |
```

## Scheduling Tasks

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

\_\_\_\_\_

To define task scheduling mode in Gantt   <b>Property:</b> <code>taskSchedulingMode</code>    <ej-gantt [taskSchedulingMode]= "taskSchedulingMode"> </ej-gantt>  TS  this.taskSchedulingMode = ej.Gantt.TaskSchedulingMode.Auto;   Not applicable
---

| To map task scheduling mode from data source | **Property:** `taskSchedulingModeMapping` <br/><br/>  
<ej-gantt taskSchedulingModeMapping= "taskMode"><br/></ej-gantt> | Not applicable |



| To enable schedule date validation while task predecessor draw action | **Property:**

`validateManualTasksOnLinking` <br/> <br/> <ej-gantt

[`validateManualTasksOnLinking`]=“true”><br/></ej-gantt> | Not applicable |

| To define working time range of day | **Property:** `dayWorkingTime` <br/><br/> <ej-gantt

[`dayWorkingTime`]=“dayWorkingTime”><br/></ej-gantt><br> **TS** <br>this.dayWorkingTime = [ {

“from”: “08:00 AM”, “to”: “12:00 PM” }]; | **Property:** `dayWorkingTime` <br/><br/> <ejs-gantt

[`dayWorkingTime`]=“dayWorkingTime”><br/></ejs-gantt><br> **TS** <br>this.dayWorkingTime = [ {

from: 9, to: 18 } ];

| To enable rounding off date value in taskbar editing | **Property:** `roundOffDayworkingTime` <br/> <br/>

<ej-gantt [`roundOffDayworkingTime`]=“true”><br/></ej-gantt> | Not applicable |

| To define non-working background color | **Property:** `nonWorkingBackground` <br/> <br/> <ej-gantt

`nonWorkingBackground` = “#0000FF”><br/></ej-gantt> | Not Applicable |

| To highlight non working time range in Gantt | **Property:** `highlightNonWorkingTime` <br/> <br/> <ej-

gantt [`highlightNonWorkingTime`]=“true”><br/></ej-gantt> | Not Applicable |

To set working days of a week | **Property:** `workweek` <br/><br/> <ej-gantt [`workweek`]=

“workweek”><br/></ej-gantt><br> **TS** <br>this.workweek =

[“Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”]; | **Property:** `workWeek` <br/><br/> <ejs-

gantt [`workWeek`]=“workWeek”><br/></ejs-gantt><br> **TS** <br>this.workWeek =

[“Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”]; |

| To enable/disable Unscheduled tasks | **Property:** `allowUnscheduledTask` <br><br> <ej-gantt

[`allowUnscheduledTask`]=“true”><br/></ej-gantt> | **Property:** `allowUnscheduledTasks` <br><br> <ejs-

gantt [`allowUnscheduledTasks`]=“true”><br/></ej-gantt> |

## Appearance and Customizations

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To define taskbar background type in Gantt | **Property:** `taskbarBackground` <br/><br/> <ej-gantt

`taskbarBackground` = “#F2F2F2”><br/></ej-gantt> | Not applicable |

| To define background color for parent taskbar | **Property:** `parentTaskbarBackground` <br/><br/> <ej-

gantt [`parentTaskbarBackground`]= “#F2F2F2”><br/></ej-gantt> | Not applicable |

| To add Taskbar height | **Property:** `taskbarHeight` <br><br> <ej-gantt [`taskbarHeight`]=“25”

[`rowHeight`]=“40”><br/></ej-gantt> | **Property:** `taskbarHeight` <br><br> <ejs-gantt

[`taskbarHeight`]=“50” [`rowHeight`]=“60”><br/></ejs-gantt> |

| To define background color for parent progress bar | **Property:** `parentProgressbarBackground`

<br/><br/> <ej-gantt [`parentProgressbarBackground`]= “#F2F2F2”><br/></ej-gantt> | Not applicable

|

| To define background color fro progress bar | **Property:** `progressbarBackground` <br/> <br/> <ej-

gantt [`progressbarBackground`]= “#F2F2F2”><br/></ej-gantt> | Not Applicable |

| To define height for progress bar | **Property:** `progressbarHeight` <br/> <br/> <ej-gantt

`progressbarHeight`= “100”><br/></ej-gantt> | Not Applicable |

| To render progress status taskbar | **Property:** *showProgressStatus* <br><br> <ej-gantt  
[showProgressStatus] = "true"><br></ej-gantt> | **Property:** *labelSettings.taskLabel* <br><br> <ejs-  
gantt [labelSettings]="labelSettings"><br></ejs-gantt><br> **TS** <br>this.labelSettings = {  
taskLabel: '\$\${Progress}%' };|

| To set connectorline width | **Property:** *connectorlineWidth* <br><br> <ej-gantt  
[connectorlineWidth]= 2><br></ej-gantt> | **Property:** *connectorLineWidth* <br><br> <ejs-gantt  
[connectorLineWidth]='3'><br></ejs-gantt>|

| To set connectorline background | **Property:** *connectorLineBackground* <br><br> <ej-gantt  
connectorLineBackground="#F2F2F2"><br></ej-gantt> | **Property:** *connectorLineBackground*  
<br><br> <ejs-gantt connectorLineBackground="red"><br></ejs-gantt> |

| To define weekend background in Gantt | **Property:** *weekendBackground* <br> <br> <ej-gantt  
weekendBackground = "blue"><br></ej-gantt> | Not applicable |

| To define taskbar template | **Property:** *taskbarTemplate* <br><br> <ej-gantt taskbarTemplate=  
"#TaskbarTemplate"><br></ej-gantt> | **Property:** *taskbarTemplate* <br><br> <ejs-gantt  
taskbarTemplate="#TaskbarTemplate"><br></ejs-gantt> |

| To define parent taskbar template | **Property:** *parentTaskbarTemplate* <br><br> <ej-gantt  
parentTaskbarTemplate= "#parentTaskbarTemplate"><br></ej-gantt> | **Property:**  
*parentTaskbarTemplate* <br><br> <ejs-gantt parentTaskbarTemplate=  
"#parentTaskbarTemplate"><br></ejs-gantt> |

| To define milestone template | **Property:** *milestoneTemplate* <br><br> <ej-gantt  
milestoneTemplate= "#milestoneTemplate"><br></ej-gantt> | **Property:** *milestoneTemplate*  
<br><br> <ejs-gantt milestoneTemplate= '#MilestoneTemplate'><br></ejs-gantt> |

| To define right task label | **Property:** *rightTaskLabelMapping* <br><br> <ej-gantt  
rightTaskLabelMapping = "taskName"><br></ej-gantt> | **Property:** *labelSettings.rightLabel*  
<br><br> <ejs-gantt [labelSettings]="labelSettings"><br></ejs-gantt><br> **TS**  
<br>this.labelSettings = { rightLabel: '\$\${taskData.Progress}' };|

| To define left task label | **Property:** *leftTaskLabelMapping* <br><br> <ej-gantt  
leftTaskLabelMapping= "taskId"><br></ej-gantt> | **Property:** *labelSettings.leftLabel* <br><br> <ejs-  
gantt [labelSettings]="labelSettings"><br></ejs-gantt><br> **TS** <br>this.labelSettings = { leftLabel:  
'TaskID' };|

| To define right task label template | **Property:** *rightTaskLabelTemplate* <br><br> <ej-gantt  
rightTaskLabelTemplate = "#customTaskRightLabel"><br></ej-gantt> | **Property:**  
*labelSettings.rightLabel* <br><br> <ejs-gantt [labelSettings]="labelSettings"><br></ejs-gantt><br>  
**TS** <br>this.labelSettings = { rightLabel: '#rightLabel' };|

| To define left task label template | **Property:** *leftTaskLabelTemplate* <br><br> <ej-gantt  
leftTaskLabelTemplate= "#customTaskLeftLabel"><br></ej-gantt> | **Property:**  
*labelSettings.leftLabel* <br><br> <ejs-gantt [labelSettings]="labelSettings"><br></ejs-gantt><br>  
**TS** <br>this.labelSettings = { leftLabel: '#leftLabel' };|



| To render resource names right to taskbar | **Property:** *showResourceNames* <br/><br/> <ej-gantt [showResourceNames]="true"><br/></ej-gantt> | **Property:** *labelSettings.rightLabel* <br/><br/> <ejs-gantt [labelSettings]="labelSettings"><br/></ejs-gantt> <br> **TS** <br>this.labelSettings = { rightLabel: 'resourceInfo'}; |

| To render task name left to taskbar | **Property:** *showTaskNames* <br/><br/> <ej-gantt [showTaskNames]="true"><br/></ej-gantt> | **Property:** *labelSettings.leftLabel* <br/><br/> <ejs-gantt [labelSettings]="labelSettings"><br/></ejs-gantt> <br> **TS** <br>this.labelSettings = { leftLabel: 'taskName'}; |

| Triggers on taskbar rendering action | **Event:** *queryTaskbarInfo* <br/><br/> <ej-gantt id="Gantt" (queryTaskbarInfo)="queryTaskbarInfo(\$event)"><br/></ej-gantt> <br> **TS** <br>queryTaskbarInfo(event) { } | **Event:** *queryTaskbarInfo* <br/><br/> <ejs-gantt (queryTaskbarInfo)="queryTaskbarInfo(\$event)"><br/></ejs-gantt> <br> **TS** <br>public queryTaskbarInfo(event) { } |

| Triggers on grid cell rendering action | **Event:** *queryCellInfo* <br/><br/> <ej-gantt id="Gantt" (queryCellInfo)="queryCellInfo(\$event)"><br/></ej-gantt> <br> **TS** <br>queryCellInfo(event) { } | **Event:** *queryCellInfo* <br/><br/> <ejs-gantt (queryCellInfo)="queryCellInfo(\$event)"><br/></ejs-gantt> <br> **TS** <br>public queryCellInfo(event) { } |

### Stripline

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To define striplines | **Property:** *stripLines* <br><br> <ej-gantt [stripLines]="stripLines"><br></ej-gantt> <br> **TS** <br>this.stripLines = [ { day: "01/02/2014", label: "Project Release", lineStyle: "dotted", lineColor: "blue", lineWidth: 2 } ]; | **Property:** *eventMarkers* <br><br> <ejs-gantt [eventMarkers]="eventMarkers"><br></ejs-gantt> <br> **TS** <br>this.eventMarkers = [ { day: '04/10/2019', cssClass: 'e-custom-event-marker', label: 'Project approval and kick-off' } ]; |

### Holidays

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To define holidays | **Property:** *holidays* <br><br> <ej-gantt [holidays]="holidays"><br></ej-gantt> <br> **TS** <br>this.holidays = [ { day: "2/03/2014", label: "Public holiday", background: "yellow" } ]; | **Property:** *holidays* <br><br> <ejs-gantt [holidays]="holidays"><br></ejs-gantt> <br> **TS** <br>this.holidays = [ { from: "04/04/2019", to: "04/05/2019", label: "Public holidays", cssClass: "e-custom-holiday" } ]; |

| To define days in holiday collection | **Property:** *holidays.day* <br/><br/> <ejs-gantt [holidays]="holidays"><br/></ejs-gantt> <br> **TS** <br>this.holidays = [ { day: "12/2/2000" } ]; | **Property:** *holidays.from* <br/><br/> <ejs-gantt [holidays]="holidays"><br/></ejs-gantt> <br> **TS** <br>this.holidays = [ { from: "3/20/2018" } ]; |

### Others

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| To define height for Gantt | **Property:** *sizeSettings.height* <br/><br/> <ej-gantt [sizeSettings]=  
"sizeSettings"><br/></ej-gantt><br/> **TS** <br/>this.sizeSettings = { height: "450px" };| **Property:**  
*height* <br/><br/><ejs-gantt height="430px"><br/></ejs-gantt>|

| To define width for Gantt | **Property:** *sizeSettings.width* <br/><br/> <ej-gantt [sizeSettings]=  
"sizeSettings"><br/></ej-gantt><br/> **TS** <br/>this.sizeSettings = { width: "700px" }><br/></ej-gantt>|  
**Property:** *width* <br/><br/><ejs-gantt width="700px"><br/></ejs-gantt>|

| To change splitter position | **Property:** *splitterPosition* <br/> <br/> <ej-gantt [splitterSettings]=  
"splitterSettings"[splitterPosition] = "50%"><br/></ej-gantt>| Not applicable |

| To change splitter by position | **Property:** *splitterSettings.position* <br/> <br/> <ej-gantt  
[splitterSettings]= "splitterSettings"><br/></ej-gantt><br/> **TS** <br/>this.splitterSettings = { position:  
"50%" };| **Property:** *splitterSettings.position* <br/> <br/> <ejs-gantt  
[splitterSettings]="splitterSettings"><br/></ejs-gantt><br/> **TS** <br/>this.splitterSettings = {  
position: "50%" };|

| To change splitter by index | **Property:** *splitterSettings.index* <br/> <br/> <ej-gantt  
[splitterSettings]= "splitterSettings"><br/></ej-gantt><br/> **TS** <br/>this.splitterSettings = { index :  
"3" };| **Property:** *splitterSettings.columnIndex* <br/> <br/> <ejs-gantt  
[splitterSettings]="splitterSettings"><br/></ejs-gantt><br/> **TS** <br/>this.splitterSettings = {  
columnIndex : "3" };|

| To define view type of Gantt | **Property:** *viewType* <br/> <br/> <ej-gantt [viewType]=  
"viewType"><br/></ej-gantt><br/> **TS** <br/>this.viewType =  
ej.Gantt.ViewType.ProjectView><br/></ej-gantt>| Not applicable |

| To enable Localization | **Property:** *locale* <br/><br/> <ej-gantt locale="fr-FR"><br/></ej-gantt>|  
**Property:** *locale* <br/><br/><ejs-gantt locale="fr-FR"><br/></ejs-gantt> |

| To specify the date format for Gantt | **Property:** *dateFormat* <br/><br/> <ej-gantt dateFormat=  
"dd/MM/yyyy"><br/></ej-gantt>| **Property:** *dateFormat* <br/><br/><ejs-gantt dateFormat=  
"dd/MM/yyyy"><br/></ejs-gantt> |

| To enable/disable key navigation | **Property:** *allowKeyboardNavigation* <br/> <br/> <ej-gantt  
[allowKeyboardNavigation]="true"><br/></ej-gantt>| **Property:** *allowKeyboard* <br/> <br/> <ejs-  
gantt [allowKeyboard]="true" ><br/></ejs-gantt>|

| To enable serial number support | **Property:** *enableSerialNumber* <br/> <br/> <ej-gantt  
[enableSerialNumber]="true"><br/></ej-gantt>| Not applicable |

| To enable/disable predecessor validation | **Property:** *enablePredecessorValidation* <br/><br/> <ej-gantt  
[enablePredecessorValidation]="true"><br/></ej-gantt>| **Property:** *enablePredecessorValidation*  
<br/><br/> <ejs-gantt [enablePredecessorValidation]="true"><br/></ejs-gantt>|

| To set timescale for working hours | **Property:** *workingTimeScale* <br/><br/> <ej-gantt  
[workingTimeScale]= "workingTimeScale"><br/></ej-gantt><br/> **TS** <br/>this.workingTimeScale =  
ej.Gantt.workingTimeScale.TimeScale24Hours;| **Property:** *dayWorkingTime* <br/><br/><ejs-gantt

[dayWorkingTime]="dayWorkingTime"></ej-gantt><br> **TS** <br>this.dayWorkingTime = [ {  
from: 0, to: 24 } ];|

| To enable work break down structure in Gantt | **Property:** *enableWBS* <br/> <br/> <ej-gantt  
[enableWBS] = "true"></ej-gantt> | Not Applicable |

| To enable work break down structure predecessor in Gantt | **Property:** *enableWBSPredecessor* <br/>  
<br/> <ej-gantt [enableWBSPredecessor] = "true"></ej-gantt> | Not Applicable |

| To map work value from data source | **Property:** *workMapping* <br/> <br/> <ej-gantt workMapping  
= "estimatedHours"></ej-gantt> | Not applicable |

| To define work unit for tasks | **Property:** *workUnit* <br/> <br/> <ej-gantt [workUnit]=  
"workUnit"></ej-gantt><br> **TS** <br>this.workUnit = ej.Gantt.WorkUnit.Day; | Not applicable |

| To define task type in Gantt | **Property:** *taskType* <br/><br/> <ej-gantt [taskType]=  
"taskType"></ej-gantt><br> **TS** <br>this.taskType = ej.Gantt.TaskType.FixedWork; | Not  
applicable |

| To enable/disable multiple exporting option | **Property:** *allowMultipleExporting* <br/><br/> <ej-gantt  
[allowMultipleExporting] = "true"></ej-gantt> | Not applicable |

| To enable virtual rendering in Gantt | **Property:** *enableVirtualization* <br/> <br/> <ej-gantt  
[enableVirtualization] = "true"></ej-gantt> | Not Applicable |

| Change splitter position dynamically | **Method:** *setSplitterIndex(index) setSplitterPosition(width)*  
<br><br> export class AppComponent {<br>constructor() {<br>&#160;&#160;...<br>}<br>public  
setSplitterIndex(event) {<br>&#160;&#160;var ganttObj =  
\$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;ganttObj.setSplitterIndex(3);<br><br>this.gan  
ttObj.ejGantt( "setSplitterPosition", "40%" );<br> | **Method:** *setSplitterPosition(value,type)* <br><br>  
@ViewChild( 'gantt' )<br>public ganttObj: GanttComponent;  
<br>this.ganttObj.setSplitterPosition( '40%', 'position' );  
<br><br>this.ganttObj.setSplitterPosition(3, 'columnIndex'); |

| To destroy Gantt | **Method:** *destroy()* <br><br> export class AppComponent {<br>constructor()  
{<br>&#160;&#160;...<br>}<br>public destroy(event) {<br>&#160;&#160;var ganttObj =  
\$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;ganttObj.destroy();<br><br>} | **Method:**  
*destroy()* <br><br> @ViewChild( 'gantt' )<br>public ganttObj: GanttComponent;  
<br>this.ganttObj.destroy(); |

| To update task id | **Method:** *updateTaskId(currentId, newId)* <br><br> export class AppComponent  
{<br>constructor() {<br>&#160;&#160;...<br>}<br>public updateTaskId(event)  
{<br>&#160;&#160;var ganttObj =  
\$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;ganttObj.updateTaskId(5, 7);<br><br>}} | Not  
applicable |

| To set scroll top for Gantt | **Method:** *setScrollTop(top)* <br><br> export class AppComponent  
{<br>constructor() {<br>&#160;&#160;...<br>}<br>public updateTaskId(event)  
{<br>&#160;&#160;var ganttObj =  
\$( "#Gantt" ).ejGantt( "instance" );<br>&#160;&#160;ganttObj.setScrollTop(50);<br><br>}} |

```
Method: setScrollTop() <br><br> @ViewChild('gantt')<br>public ganttObj: GanttComponent;
<br>this.ganttObj.setScrollTop(200); |
```

```
| To get columns to edit in resource view | Method: getResourceViewEditColumns() <br><br> export  
class AppComponent {<br>constructor() {<br>#160;#160;//...<br>}<br>public  
getResourceViewEditColumns(event) {<br>#160;#160;var ganttObj =  
$("#Gantt").ejGantt("instance");<br>#160;#160;columns =  
ganttObj.getResourceViewEditColumns() | Not applicable |
```

```
| Show/hide critical path in Gantt | Method: showCriticalPath(isShown) <br/><br/>export class  
AppComponent {<br>constructor() {<br>#160;#160;//...<br>}<br>public  
showCriticalPath(event) {<br>#160;#160;var ganttObj =  
$(("#Gantt").ejGantt("instance");<br>#160;#160;ganttObj.showCriticalPath(true);<br>}<br>  
| Not applicable |
```

```
|Triggers on initialization of Gantt control | Event: load <br/><br/><ej-gantt id="Gantt"
(load)="load($event)"><br/></ej-gantt><br> TS <br>load(event) { } | Event: load <br/><br/><ej-
gantt (load)="load($event)"><br/></ej-gantt><br> TS <br>public load(event) { } |
```

```
| Triggers after splitter resize action | Event: splitterResized <br><br><ej-gantt id="Gantt"
(splitterResized)="splitterResized($event)"><br></ej-gantt><br> TS <br>splitterResized(event) { }
| Event: splitterResized <br><br><ejs-gantt
(splitterResized)="splitterResized($event)"><br></ejs-gantt><br> TS <br>public
splitterResized(event) { } |
```

Triggers when taskbar item is clicked   <b>Event:</b> <code>taskbarClick</code> (taskbarClick)="taskbarClick(\$event)" TS <code>taskbarClick(event) { }</code>   Not applicable
---

## Style and appearance in Angular Gantt component

To modify the Gantt Chart appearance, you need to override the default CSS of gantt chart. Please find the list of CSS classes and its corresponding section in Gantt Chart. Also, you have an option to create your own custom theme for all the JavaScript controls using our [Theme Studio](#).

Section | CSS Class | Purpose of Class

**Root** | e-gantt | This class is in the root element (div) of the gantt chart control.

**Header** | e-gridheader | This class is added in the root element of header element. In this class, You can override thin line between header and content of the gantt chart.

e-table	This class is added at 'table' of the gantt chart header. This CSS class makes table width as 100%.
---------	---

Class	Description
<code> e-columnheader </code>	This class is added at 'tr' of the gantt chart header.

**Grid Content**|e-gridcontent|This class is added at root of body content. This is to override background color of the body.

**|| e-table|**This class is added to table of content. This CSS class makes table width as 100 %.

||e=row|This class is added to rows of gantt chart.

**|e-altrow** | This class is added to alternate rows of gantt chart. This is to override alternate row color of the gantt chart.

**|e-rowcell** | This class is added to all cells in the gantt chart. This is to override cells appearance and styling.

**Chart Content** |e-gantt-chart| This class is added to the chart side of the gantt chart.

**|e-chart-row** | This class is added to rows of gantt chart.

**Timeline** |e-timeline-header-container| This class is added to timeline of the gantt chart.

**|e-header-cell-label** | This class is added to the header cell of the timeline.

**|e-weekend-header-cell** | This class is added to the weekend cells.

**Taskbar** |e-taskbar-main-container| This class is added to taskbar of the gantt chart.

**|e-gantt-parent-taskbar** | This class is added to the parent task bar of the gantt chart.

**|e-gantt-milestone** | This class is added to the milestone tasks of the gantt chart.

**|e-gantt-unscheduled-taskbar** | This class is added to the unscheduled tasks.

**|e-gantt-manualparenttaskbar** | This class is added to the manual scheduled parent taskbar.

**|e-gantt-child-manualtaskbar** | This class is added to the manual scheduled child taskbar.

**|e-gantt-unscheduled-manualtask** | This class is added to the manual unscheduled tasks.

**Splitter** |e-split-bar| This class is added to the gantt chart splitter.

**|e-resize-handler** | This class is added to the resize handler of the gantt chart splitter.

**|e-arrow-left** | This class is added to the left arrow of the resize handler.

**|e-arrow=right** | This class is added to the right arrow of the resize handler.

**Connector Lines** |e-line| This class is added to the connector lines.

**|e-connector-line-right-arrow** | This class is added to the right arrow of the connector line.

**|e-connector-line-left-arrow** | This class is added to the left arrow of the connector line.

**Labels** |e-task-label| This class is added to the task labels.

**|e-right-label-container** | This class is added to the right label.

**|e-left-label-container** | This class is added to the left label.

**Event Markers** |e-event-markers| This class is added to the event markers.

**Baseline** |e-baseline-bar| This class is added to the baseline.

**|e-baseline-gantt-milestone-container** | This class is added to the baseline of milestone tasks.

**Tooltip** |e-gantt-tooltip| This class is added to the tooltip.

### Grid lines

In Gantt component, you can show or hide the grid lines in the TreeGrid side and chart side by using the [gridLines](#) property.

The following options are available in Gantt component for rendering grid lines,

- Horizontal: The horizontal grid lines alone will be visible.
- Vertical: The vertical grid lines alone will be visible.
- Both: Both the horizontal and vertical grid lines will be visible on the TreeGrid and chart sides.
- None: Gridlines will not be visible on TreeGrid and chart sides.

By default, the [gridLines](#) property is set with **Horizontal** type.

The following code example shows how to change the gridlines rendering mode in the Gantt component.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt id="ganttDefault" height="430px" [dataSource]="data"
    [taskFields]="taskSettings" [gridLines] = "gridLines"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public gridLines?: string;
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.gridLines = 'Both';
  }
}
```

#### 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 add edit dialog in Angular Gantt component

In the Gantt component, add and edit dialogs can be opened dynamically by using [openAddDialog](#) and [openEditDialog](#) methods. The following code example shows how to open add and dialog on separate button click actions.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { EditSettingsModel } from '@syncfusion/ej2-angular-gantt';
import { editingData, editingResources } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='editDialog' (click)='edit()'>Edit
Dialog</button>
    <br><br><br>
    <button ej2-button id='addDialog' (click)='add()'>Add Dialog</button>
    <br><br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"
[editDialogFields]="editDialogFields" [editSettings]="editSettings"
[resourceNameMapping]= "resourceNameMapping"
[resourceIdMapping]="resourceIdMapping" [resources]= "resources"></ejs-
gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public editDialogFields?: object[];
  public resourceNameMapping?: string;
  public resourceIdMapping?: string;
  public resources?: object[];
  public labelSettings?: object;
  public editSettings?: EditSettingsModel;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
```

```

        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
        notes: 'info',
        resourceInfo: 'resources'
    };
    this.editDialogFields = [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' },
        { type: 'Resources' },
        { type: 'Notes' }
    ];
    this.resourceNameMapping = 'resourceName';
    this.resourceIdMapping = 'resourceId';
    this.resources = editingResources;
    this.editSettings = {
        allowEditing: true,
        allowTaskbarEditing: true
    };
}
edit(): void {

    this.ganttObj.editModule.dialogModule.openEditDialog(this.ganttObj.selectedRowIndex);
};
add(): void {
    this.ganttObj.editModule.dialogModule.openAddDialog();
};
}

```

### MAIN.TS

```

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

```

NOTE: You should select any one of the row in the Gantt to open the edit dialog.

### SetScrollTop in Angular Gantt component

In the Gantt component, you can set the vertical scroller position dynamically by clicking the custom button using the [setScrollTop](#) method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';

```



```

import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='scrolltop' (click)='scroll()'>Set Scroll
Top</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"
[splitterSettings]="splitterSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public splitterSettings?: object;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.splitterSettings = {
      position: "50%"
    };
  }
  scroll(): void {
    this.ganttObj.ganttChartModule.scrollObject.setScrollTop(500);
  }
}

```

## MAIN.TS

```

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

```

## Change schedule dates in Angular Gantt component

In the Gantt component, you can change the schedule start and end dates by clicking the custom button programmatically using the [updateProjectDates](#) method. You can pass the start and end dates as method arguments to the [updateProjectDates](#) method. You can also pass the Boolean value as an

additional parameter, which is used to round-off the schedule start and end dates displayed in Gantt timeline.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='changedate' (click)='change()'>Change
Date</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent;
  public ngOnInit(): void {
    this.data = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
  }
  change(): void {
    this.ganttObj!.updateProjectDates(new Date('04/10/2019'), new
Date('06/20/2019'), true);
  }
}
```

### MAIN.TS

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

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

### Copy paste records in Angular Gantt component

You can copy and paste a record in the Gantt chart by using the `addRecord` method and `custom context menu`. It is also possible to copy and paste the parent record with multiple hierarchical child records on the required position.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule, ContextMenuService, EditService, SelectionService }
from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent, ContextMenuClickEventArgs, ContextMenuOpenEventArgs
} from '@syncfusion/ej2-angular-gantt';
import { ContextMenuItemModel } from '@syncfusion/ej2-grids';
import { editingData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [SelectionService, ContextMenuService, EditService],
  standalone: true,
  selector: 'app-root',
  template:
`<ejs-gantt #customcontextmenu id="ganttCustomContextMenu"
height="430px" [dataSource]="editingData" [taskFields]="taskSettings"
[enableContextMenu]="true" [contextMenuItems]="contextMenuItems"
[editSettings]="editSettings" (contextMenuClick)="contextMenuClick($event)"
(contextMenuOpen)="contextMenuOpen($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public copiedRecord?: any;
  public editingData?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public contextMenuItems?: (string | ContextMenuItemModel)[];
  @ViewChild('customcontextmenu', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.editingData = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.editSettings = {
```

```

        allowAdding: true,
        allowEditing: true,
        allowDeleting: true
    };
    this.contextMenuItems = [{ text: 'Copy', target: '.e-content', id:
'copy' } as ContextMenuItemModel,
        { text: 'Paste', target: '.e-content', id: 'paste' } as
ContextMenuItemModel,
    ];
    }
    public contextMenuClick (args: ContextMenuClickEventArgs) {
        if (args.item.id === 'copy') {
            this.copiedRecord = args.rowData;
            this.copiedRecord.taskData.TaskID =
this.ganttObj.currentViewData.length + 1;
        }
        if (args.item.id === 'paste') {

this.ganttObj.addRecord(this.copiedRecord.taskData, 'Below', args.rowData!.ind
ex);

            if(this.copiedRecord.hasChildRecords) {
                addChildRecords(this.copiedRecord, args.rowData!.index! +
1);
            }
            this.copiedRecord = undefined;
        }
    }
    public contextMenuOpen (args: ContextMenuOpenEventArgs) {
        if (args.type !== 'Header') {
            if (this.copiedRecord) {
                args.hideItems!.push('Copy');
            } else {
                args.hideItems!.push('Paste');
            }
        }
    }
}
function addChildRecords(this: any, record: any, index: any): void {
    for(var i=0; i<record.childRecords.length; i++) {
        var childRecord = record.childRecords[i];
        childRecord.taskData.TaskID = this.ganttObj.currentViewData.length
+ 1;

        this.ganttObj.addRecord(childRecord.taskData, 'Child', index);
        if(childRecord.hasChildRecords) {
            addChildRecords(childRecord, index + (i+1));
        }
    }
}
}

```

## MAIN.TS

```

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

```

### Maintain record index in Angular Gantt component

Row dropped record's index position can be maintained in the Gantt chart by changing the database table index position using the `rowDrop` event. In this event, the `fromIndex` and `dropIndex` values can be passed to the server side using Ajax request. On the server side, the `insert` and `insertAtTop` methods are used to update the row index position. The following code snippets explain the solution.

`typescript

```
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { ganttData } from './data';
import { Gantt } from '@syncfusion/ej2-gantt';
@Component({
  selector: 'app-root',
  template:
    `
```

```

};
}
public rowDrop: function (args) {
    public record = this.flatData[args.fromIndex][this.taskFields.id];
    public record2 = this.flatData[args.dropIndex][this.taskFields.id];
    public data: IGanttData = args.data[0];
    public uri = 'https://localhost:44379/Home/RowDropMethod';
    public dragdropdata = {
        record: data[0].taskData,
        position: args.dropIndex,
        dragidMapping: record,
        dropidMapping: record2
    };
    public ajax = new Ajax(
    {
        url: uri,
        type: 'POST',
        contentType: "application/json",
        data: JSON.stringify(dragdropdata),
    });
    ajax.send();
};
}
`
`typescript
public IActionResult RowDropMethod([FromBody]DragDropData value)
{
    var data = new CRUDModel();
    copyRecord = true;
    if (value.position == "bottomSegment" || value.position == "topSegment")
    {
        {
            var childCount = 0;

```

```
int parent1 = (int)value.record.parentID;
childCount += FindChildRecords(parent1);
if (childCount == 1)
{
    var i = 0;
    for (; i < DataList.Count; i++)
    {
        if (DataList[i].taskID == parent1)
        {
            DataList[i].isParent = false;
            break;
        }
        if (DataList[i].taskID == value.record.taskID)
        {
            DataList[i].parentID = null;
            break;
        }
    }
}

DataList.Remove(DataList.Where(ds => ds.taskID == value.dragidMapping).FirstOrDefault());
var j = 0;
for (; j < DataList.Count; j++)
{
    if (DataList[j].taskID == value.dropidMapping)
    {
        value.record.parentID = DataList[j].parentID;
        break;
    }
}

data.Value = value.record;
if (value.position == "bottomSegment")
{
```

```

this.Insert(data, value.dropidMapping);
}
else if (value.position == "topSegment")
{
this.InsertAtTop(data, value.dropidMapping);
}
}
else if (value.position == "middleSegment")
{
DataList.Remove(DataList.Where(ds => ds.taskID == value.dragidMapping).FirstOrDefault());
value.record.parentID = value.dropidMapping;
FindDropdata(value.dropidMapping);
data.Value = value.record;
this.Insert(data, value.dropidMapping);
}
copyRecord = false;
return Json(new { updatedRecords = value.record });
}
,

```

### Custom field in Angular Gantt component

Generally in Gantt, Custom fields are displayed in the Custom Tab of the Add/Edit dialogs. However, they can be included in the General Tab of Add/Edit Dialog Box using `actionBegin` and `actionComplete` events. These events are used to append the custom field to the dialog box. The following code snippets demonstrate the solution.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService, ToolbarService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent, ToolbarItem, EditSettingsModel,
 SelectionSettingsModel } from '@syncfusion/ej2-angular-gantt';
import { CheckBox } from "@syncfusion/ej2-buttons";
import { TextBox, NumericTextBox, MaskedTextBox } from "@syncfusion/ej2-
inputs";
import { DatePicker, DateTimePicker } from "@syncfusion/ej2-calendars";
import { DropDownList } from "@syncfusion/ej2-dropdowns";
import { editingData } from '../data';

```



```

@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<ejs-gantt #gantt id="ganttDefault" height="430px"
    [dataSource]="editingData" [taskFields]="taskSettings" [toolbar]="toolbar"
    [editDialogFields]="editDialogFields" [addDialogFields]="addDialogFields"
    [editSettings]="editSettings" [columns]="columns"
    (actionBegin)="$event"
    (actionComplete)="$event"> </ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public divElement?: any;
  public inputs = {
    booleanedit: CheckBox,
    dropdownedit: DropDownList,
    datepickeredit: DatePicker,
    datetimepickeredit: DateTimePicker,
    maskededit: MaskedTextBox,
    numericedit: NumericTextBox,
    stringedit: TextBox
  };
  public editingData?: object[];
  public taskSettings?: object;
  public editSettings?: object;
  public editDialogFields?: object[];
  public addDialogFields?: object[];
  public columns?: object[];
  public toolbar?: ToolbarItem[];
  public selectionSettings?: SelectionSettingsModel;
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.editingData = editingData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      child: 'subtasks'
    };
    this.editSettings = {
      allowEditing: true,
      allowAdding: true,
      allowDeleting: true,
      mode: "Dialog"
    };
    this.editDialogFields = [
      { type: 'General', headerText: 'General' },

```

```

        { type: 'Dependency' },
        { type: 'Resources' },
        { type: 'Notes' }
    ];
    this.addDialogFields = [
        { type: 'General', headerText: 'General' },
        { type: 'Dependency' },
        { type: 'Resources' },
        { type: 'Notes' }
    ];
    this.columns = [
        { field: 'TaskID', headerText: 'Task ID', textAlign: 'Left',
width: '100' },
        { field: 'TaskName', headerText: 'Task Name', width: '250' },
        { field: 'StartDate', headerText: 'Start Date', width: '150' },
        { field: 'EndDate', headerText: 'End Date', width: '150' },
        { field: 'Duration', headerText: 'Duration', width: '150' },
        { field: 'Progress', headerText: 'Progress', width: '150' },
        { field: 'CustomField', headerText: 'CustomField', width: '150' }
    ]
    };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
'ExpandAll', 'CollapseAll'];
    }
    public actionBegin(args: any) {
        if (args.requestType === "beforeOpenEditDialog" || args.requestType
=== "beforeOpenAddDialog" ) {
            var column = this.ganttObj.columnByField("CustomField");
            this.divElement = this.ganttObj.createElement("div", { className:
"e-edit-form-column" });
            var inputElement: any;
            inputElement = this.ganttObj.createElement("input", {
                attrs: {
                    type: "text",
                    id: this.ganttObj.controlId + "" + column.field,
                    name: column.field,
                    title: column.field
                }
            });
            this.divElement.appendChild(inputElement);
            var input = {
                enabled: true,
                floatLabelType: "Auto",
                placeholder: "CustomField",
                value: args.rowData.CustomField
            };
            var inputObj = new (this as any).inputs[column.editType](input);
            inputObj.appendTo(inputElement);
        }
    };
    public actionComplete(args: any) {
        if (args.requestType === "openEditDialog" || args.requestType ===
"openAddDialog" ) {
            var generalTab = document.getElementById(this.ganttObj.controlId +
"GeneralTabContainer");
            generalTab!.appendChild(this.divElement);
        }
    }

```

```
};
}
```

## MAIN.TS

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

### Maintain zoom to fit in Angular Gantt component

In the Gantt control, While performing edit actions or dynamically change dataSource, the timeline gets refreshed. When zoomToFit toolbar item is clicked and perform editing actions or dynamically change dataSource, the timeline gets refreshed. So that, the timeline will not fit to the project any more.

#### Maintain zoomToFit after edit actions

We can maintain zoomToFit after editing actions(cell edit,dialog edit,taskbar edit) by using [fitToProject](#) method in `actionComplete` and `taskbarEdited` event.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { EditService, SelectionService, ToolbarService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';
import { ToolbarItem, ZoomTimelineSettings, EditSettingsModel } from
 '@syncfusion/ej2-angular-gantt';
import { projectNewData } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `

```

```

public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks'
    };
    this.editSettings = {
        allowEditing: true,
        allowTaskbarEditing: true,
    };
    this.toolbar = ['Edit', 'ZoomToFit'];
}
public actionComplete(args: any) {
    if ((args.action === "CellEditing" || args.action === "DialogEditing")
    && args.requestType === "save") {
        this.ganttObj.dataSource = projectNewData;
        this.ganttObj.fitToProject();
    }
};
public taskbarEdited(args: any) {
    if (args) {
        this.ganttObj.dataSource = projectNewData;
        this.ganttObj.fitToProject();
    }
};
}

```

### MAIN.TS

```

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

```

### Maintain zoomToFit after change dataSource dynamically

We can maintain `zoomToFit` after change `dataSource` dynamically, by calling `fitToProject` method in `dataBound` event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule } from '@syncfusion/ej2-angular-gantt'
import { ToolbarService } from '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
'@angular/core';
import { Gantt } from '@syncfusion/ej2-gantt';
import { GanttComponent } from '@syncfusion/ej2-angular-gantt';

```

```

import { ToolbarItem, ZoomTimelineSettings } from '@syncfusion/ej2-angular-gantt';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { projectNewData, data } from './data';
@Component({
  imports: [
    GanttModule
  ],
  providers: [ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='changeData' (click)='changeData()'>Change
Data</button>
    <br><br>
    <ejs-gantt #gantt id="ganttDefault" height="430px"
[dataSource]="data" [taskFields]="taskSettings"[toolbar]="toolbar"
(dataBound)="dataBound($event)"></ejs-gantt>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public toolbar?: ToolbarItem[];
  @ViewChild('gantt', {static: true})
  public ganttObj?: GanttComponent | any;
  public ngOnInit(): void {
    this.data = projectNewData;
    this.taskSettings = {
      id: 'TaskID',
      name: 'TaskName',
      startDate: 'StartDate',
      endDate: 'EndDate',
      duration: 'Duration',
      progress: 'Progress',
      dependency: 'Predecessor',
      child: 'subtasks'
    };
    this.toolbar = ['ZoomToFit'];
  }
  public dataBound(args: any) {
    this.ganttObj.fitToProject();
  };
  changeData(): void {
    this.ganttObj.dataSource = data;
  };
}

```

## 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 from another component in Angular Gantt component

In Gantt, it is possible to drag a record from another component and drop it in Gantt chart with updating the Gantt record. Here, dragging an item from TreeView component to Gantt and that item is updated as a resource for the Gantt record, we can achieve this, by using [nodeDragStop](#) event of TreeView control.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GanttModule, GanttAllModule } from '@syncfusion/ej2-angular-gantt'
import { TreeViewModule } from '@syncfusion/ej2-angular-navigations'
import { EditService, SelectionService, ToolbarService } from
 '@syncfusion/ej2-angular-gantt'
import { Component, ViewEncapsulation, OnInit, ViewChild } from
 '@angular/core';
import { editingData, editingResources } from './data';
import { DragAndDropEventArgs } from '@syncfusion/ej2-navigations';
import { TreeViewComponent } from '@syncfusion/ej2-angular-navigations';
import { closest, addClass } from '@syncfusion/ej2-base';
import { GanttComponent, ToolbarItem, EditSettingsModel,
 SelectionSettingsModel } from '@syncfusion/ej2-angular-gantt';
@Component({
  imports: [
    GanttModule, GanttAllModule, TreeViewModule
  ],
  providers: [EditService, SelectionService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<p><b>Gantt</b></p>
    <ejs-gantt id="ganttDefault" #gantt height="280px" [dataSource]="data"
    [taskFields]="taskSettings" [columns]="columns"
    [timelineSettings]="timelineSettings" [labelSettings]="labelSettings"
    [treeColumnIndex]="1" height="450px" [allowSelection]="true" dateFormat="MMM
    dd, y" [projectStartDate]="projectStartDate"
    [projectEndDate]="projectEndDate" [highlightWeekends]="true"
    [gridLines]="gridLines" [editSettings]="editSettings" [toolbar]="toolbar"
    [resourceFields]="resourceFields" [resources]="resources"
    [splitterSettings]="splitterSettings"></ejs-gantt>
    <p><b>List</b></p>
    <ejs-treeview id="tree1" #treeObj height="200px" [fields]='field'
    [allowDragAndDrop]='allowDragAndDrop' (nodeDragStop)="onDragStop($event)"
    ></ejs-treeview>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for Gantt
  public data?: object[];
  public taskSettings?: object;
  public columns?: object[];
  public labelSettings?: object;
  public selectionSettings?: object;
  public projectStartDate?: Date;
  public projectEndDate?: Date;
  public toolbar?: object;
```

```

public splitterSettings?: object;
public editSettings?: object;
public gridLines?: string;
public resources?: object[];
public resourceFields?: object;
public field: Object = { dataSource: editingResources, id: 'resourceId',
text: 'resourceName' };
public allowDragAndDrop: boolean = true;
@ViewChild('gantt')
public ganttObj?: GanttComponent | any;
timelineSettings: any;
public ngOnInit(): void {
    this.data = editingData;
    this.resources = editingResources;
    this.taskSettings = {
        id: 'TaskID',
        name: 'TaskName',
        startDate: 'StartDate',
        endDate: 'EndDate',
        duration: 'Duration',
        progress: 'Progress',
        dependency: 'Predecessor',
        child: 'subtasks',
        notes: 'info',
        resourceInfo: 'resources'
    };
    this.editSettings = {
        allowAdding: true,
        allowEditing: true,
        allowDeleting: true,
        allowTaskbarEditing: true,
        showDeleteConfirmDialog: true
    };
    this.selectionSettings = {
        type: 'Multiple'
    };
    this.resourceFields = {
        id: 'resourceId',
        name: 'resourceName'
    };
    this.labelSettings = {
        leftLabel: 'TaskName',
        rightLabel: 'resources'
    };
    this.projectStartDate= new Date('03/28/2019');
    this.projectEndDate= new Date('07/06/2019');
    this.splitterSettings = {
        columnIndex: 2
    };
    this.gridLines= 'Both';
    this.toolbar = [ 'Add', 'Update', 'Edit', 'Delete', 'ExpandAll',
'CollapseAll',,];
    }
    created(): void {
        addClass([this.ganttObj.ganttChartModule.chartElement], 'e-
draggable');
    };
}

```

```

onDragStop(args: DragAndDropEventArgs): void {
  let chartEle: any = closest(args.target, '.e-chart-row');
  let gridEle: any = closest(args.target, '.e-row');
  if(gridEle){
    var index = this.ganttObj.treeGrid.getRows().indexOf(gridEle);
    this.ganttObj.selectRow(index);
  }
  if (chartEle) {
    var index = chartEle.ariaRowIndex;
    this.ganttObj.selectRow(Number(index));
  }
  let record: any = args.draggedNodeData;
  let selectedData =
this.ganttObj.flatData[this.ganttObj.selectedRowIndex];
  let selectedDataResource = selectedData.taskData.resources;
  let resources = [];
  if (selectedDataResource) {
    for (var i = 0; i < selectedDataResource.length; i++) {
      resources.push(selectedDataResource[i].resourceId);
    }
  }
  resources.push(parseInt(record.id));
  if (chartEle || gridEle) {
    var data = {
      TaskID: selectedData.taskData.TaskID,
      resources: resources
    };
    this.ganttObj.updateRecordByID(data);
    var elements = document.querySelectorAll('.e-drag-item');
    while (elements.length > 0 && elements[0].parentNode) {
      elements[0].parentNode.removeChild(elements[0]);
    }
  }
}
}

```

### MAIN.TS

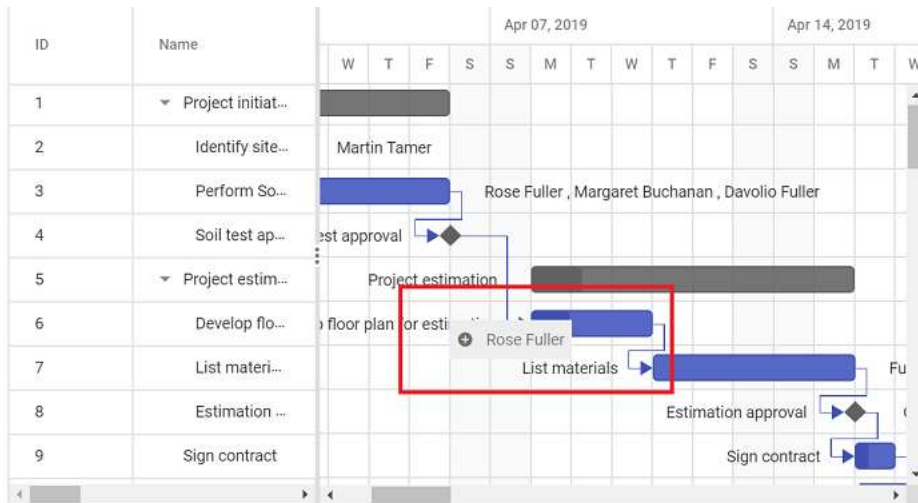
```

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

```

The following screenshot shows dropping record from another component in to Gantt, and **Rose Fuller** is added as resource for the task **Develop floor plan estimation**.





#### Flat list

Martin Tamer
Rose Fuller
Margaret Buchanan
Fuller King

[Link to the Video](#)

## Grid

### Getting started with Angular Grid component

This section explains you the steps required to create a simple Grid and demonstrate the basic usage of the Grid component in an Angular environment.

To get start quickly with Angular Grid 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@16.0.1
```

```
`
```

#### Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
```

```
ng new my-app
```

```
`
```

This command will prompt you for a few settings for the new project, such as whether to add Angular routing and which stylesheet format to use.

```
D:\Sample>ng new syncfusion-angular-app
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org ]
```

By default, it will create a CSS-based application.

Next, navigate to the created project folder:

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

### Adding Syncfusion Grid package

All the available Essential JS 2 packages are published in [npmjs.com](https://www.npmjs.com) registry.

To install Grid component, use the following command.

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

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

### Registering Grid Module

Import Grid module into Angular application(app.module.ts) from the package **@syncfusion/ej2-angular-grids** [src/app/app.module.ts].

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the GridModule for the Grid component
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-grids module into NgModule
  imports: [ BrowserModule, GridModule ],
  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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-notifications/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

,

### Add Grid component

Modify the template in `[src/app/app.component.ts]` file to render the grid component.

Add the Angular Grid by using `<ejs-grid>` selector in **template** section of the `app.component.ts` file.

```
`typescript
```

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the Grid component
  template: '<ejs-grid> </ejs-grid>'
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
```

,

### Defining Row Data

Bind data for the Grid component by using [dataSource](#) property. It accepts either array of JavaScript object or [DataManager](#) instance.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  selector: 'app-root',
  template: <ejs-grid [dataSource]='data'> </ejs-grid>
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}

```

Create a [src/app/datasource.ts] file and utilize the following dataset to provide JSON data for the grid component.

```

`typescript
export let data: Object[] = [
{
  OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date(8364186e5),
  ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
  ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
},
{
  OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date(836505e6),
  ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
  ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date(8367642e5),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
},

```

```
{
  OrderID: 10251, CustomerID: 'VICTE', EmployeeID: 3, OrderDate: new Date(8367642e5),
  ShipName: 'Victuailles en stock', ShipCity: 'Lyon', ShipAddress: '2, rue du Commerce',
  ShipRegion: 'CJ', ShipPostalCode: '69004', ShipCountry: 'France', Freight: 41.34, Verified: !0
},
{
  OrderID: 10252, CustomerID: 'SUPRD', EmployeeID: 4, OrderDate: new Date(8368506e5),
  ShipName: 'Suprêmes délices', ShipCity: 'Charleroi', ShipAddress: 'Boulevard Tirou, 255',
  ShipRegion: 'CJ', ShipPostalCode: 'B-6000', ShipCountry: 'Belgium', Freight: 51.3, Verified: !0
},
{
  OrderID: 10253, CustomerID: 'HANAR', EmployeeID: 3, OrderDate: new Date(836937e6),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 58.17, Verified: !0
},
{
  OrderID: 10254, CustomerID: 'CHOPS', EmployeeID: 5, OrderDate: new Date(8370234e5),
  ShipName: 'Chop-suey Chinese', ShipCity: 'Bern', ShipAddress: 'Hauptstr. 31',
  ShipRegion: 'CJ', ShipPostalCode: '3012', ShipCountry: 'Switzerland', Freight: 22.98, Verified: !1
},
{
  OrderID: 10255, CustomerID: 'RICSU', EmployeeID: 9, OrderDate: new Date(8371098e5),
  ShipName: 'Richter Supermarkt', ShipCity: 'Genève', ShipAddress: 'Starenweg 5',
  ShipRegion: 'CJ', ShipPostalCode: '1204', ShipCountry: 'Switzerland', Freight: 148.33, Verified: !0
},
{
  OrderID: 10256, CustomerID: 'WELLI', EmployeeID: 3, OrderDate: new Date(837369e6),
  ShipName: 'Wellington Importadora', ShipCity: 'Resende', ShipAddress: 'Rua do Mercado, 12',
  ShipRegion: 'SP', ShipPostalCode: '08737-363', ShipCountry: 'Brazil', Freight: 13.97, Verified: !1
},
{
  OrderID: 10257, CustomerID: 'HILAA', EmployeeID: 4, OrderDate: new Date(8374554e5),
```

```

ShipName: 'HILARION-Abastos', ShipCity: 'San Cristóbal', ShipAddress: 'Carrera 22 con Ave. Carlos
Soubllette #8-35',
ShipRegion: 'Táchira', ShipPostalCode: '5022', ShipCountry: 'Venezuela', Freight: 81.91, Verified: !0
},
{
OrderID: 10258, CustomerID: 'ERNSH', EmployeeID: 1, OrderDate: new Date(8375418e5),
ShipName: 'Ernst Handel', ShipCity: 'Graz', ShipAddress: 'Kirchgasse 6',
ShipRegion: 'CJ', ShipPostalCode: '8010', ShipCountry: 'Austria', Freight: 140.51, Verified: !0
},
{
OrderID: 10259, CustomerID: 'CENTC', EmployeeID: 4, OrderDate: new Date(8376282e5),
ShipName: 'Centro comercial Moctezuma', ShipCity: 'México D.F.', ShipAddress: 'Sierras de Granada
9993',
ShipRegion: 'CJ', ShipPostalCode: '05022', ShipCountry: 'Mexico', Freight: 3.25, Verified: !1
},
{
OrderID: 10260, CustomerID: 'OTTIK', EmployeeID: 4, OrderDate: new Date(8377146e5),
ShipName: 'Ottilies Käseladen', ShipCity: 'Köln', ShipAddress: 'Mehrheimerstr. 369',
ShipRegion: 'CJ', ShipPostalCode: '50739', ShipCountry: 'Germany', Freight: 55.09, Verified: !0
},
{
OrderID: 10261, CustomerID: 'QUEDE', EmployeeID: 4, OrderDate: new Date(8377146e5),
ShipName: 'Que Delícia', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua da Panificadora, 12',
ShipRegion: 'RJ', ShipPostalCode: '02389-673', ShipCountry: 'Brazil', Freight: 3.05, Verified: !1
},
{
OrderID: 10262, CustomerID: 'RATTC', EmployeeID: 8, OrderDate: new Date(8379738e5),
ShipName: 'Rattlesnake Canyon Grocery', ShipCity: 'Albuquerque', ShipAddress: '2817 Milton Dr.',
ShipRegion: 'NM', ShipPostalCode: '87110', ShipCountry: 'USA', Freight: 48.29, Verified: !0
}};
`

```

### Defining Columns

The Grid has an option to define columns as directives. In these column directives, we have properties to customize columns.

Let's check the properties used here:

- We have added [field](#) to map with a property name an array of JavaScript objects.
- We have added [headerText](#) to change the title of columns.
- We have used [textAlign](#) to change the alignment of columns.

By default, columns will be left aligned. To change columns to right align, we need to define [textAlign](#) as **Right**.

- Also, we have used another useful property, [format](#).

Using this, we can format number and date values to standard or custom formats.

Here, we have defined it for the conversion of numeric values to currency.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

**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 grid with additional features, inject the required modules. The following modules are used to extend grid's basic functionality.

- **PageService** - Inject this provider to use paging feature.
- **SortService** - Inject this provider to use sorting feature.
- **FilterService** - Inject this provider to use filtering feature.
- **GroupService** - Inject this provider to use grouping feature.

These modules should be injected into the **providers** section of root **NgModule** or component class.

Additional feature modules are available [here](#).

**Enable Paging**

The paging feature enables users to view the Grid record in a paged view. It can be enabled by setting [allowPaging](#) property to true. Also, need to inject the **PageService** module in the provider section as follows. If we didn't inject the **PageService** module, then the pager will not be rendered in Grid. The pager can be customized using [pageSettings](#) property.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
  [pageSettings]='pageSettings'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
  textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
  width=120></e-column>
```



```

        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageSettings?: PageSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.pageSettings = { pageSize: 6 };
        }
    }
}

```

### MAIN.TS

```

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

```

### Enable Sorting

The sorting feature enables the user to order the records. It can be enabled by setting [allowSorting](#) property as true. Also, need to inject the **SortService** module in the provider section as follow. If we didn't inject the **SortService** module, then user not able to sort when click on headers. Sorting feature can be customized using [sortSettings](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [PageService,
                SortService,
                FilterService,
                GroupService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true" [pageSettings]="pageSettings">
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>

```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageSettings?: PageSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.pageSettings = { pageSize: 6 };
        }
    }
}

```

### MAIN.TS

```

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

```

### Enable Filtering

The filtering feature enables the user to view the reduced amount of records based on filter criteria. It can be enabled by setting [allowFiltering](#) property as true. Also, need to inject the **FilterService** module in the provider section as follow. If we didn't inject the **FilterService** module, then filter bar will not be rendered in Grid. Filtering feature can be customized using [filterSettings](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true"
[allowFiltering]="true" [pageSettings]="pageSettings">

```

```

        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
            <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageSettings?: PageSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.pageSettings = { pageSize: 6 };
        }
    }
}

```

### 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

The grouping feature enables users to view the Grid record in a grouped view. It can be enabled by setting [allowGrouping](#) property to true. Also, need to inject the **GroupService** module in the provider section as follow. If we didn't inject the **GroupService** module, then the group drop area will not be rendered in Grid. Grouping feature can be customized using [groupSettings](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService,
ToolbarService, ExcelExportService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [PageService,
        SortService,
        FilterService,
        GroupService,
        ToolbarService,
        ExcelExportService],
})

```

```

standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true"
[allowFiltering]="true" [allowGrouping]="true"
[pageSettings]='pageSettings'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageSettings?: PageSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSize: 6 };
  }
}

```

### 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

Use the following command to run the application in browser.

```

`javascript
ng serve --open
`

```

The output will appear as follows.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService,
ToolbarService, ExcelExportService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

```

```

        GridModule

    ],
    providers: [PageService,
                SortService,
                FilterService,
                GroupService,
                ToolbarService,
                ExcelExportService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true"
                [allowFiltering]="true" [allowGrouping]="true"
[pageSettings]='pageSettings'>
                <e-columns>
                    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
                    <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                    <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
                    <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
                </e-columns>
            </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageSettings?: PageSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.pageSettings = { pageSize: 6 };
        }
    }
}

```

## 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 get current query in Angular Grid](#)
- [Apply Grid Localization from external JSON file in Angular Grid](#)
- [How to render the Syncfusion Grid in jHipster spring boot Angular App](#)
- [Calculate Height of Angular Grid](#)
- [How to set tabindex for input element and NumericTextBox in Angular Grid](#)
- [How to toggle between List and Grid View in Angular?](#)
- [How to get started easily with an example of Syncfusion angular 6 data grid/datatable?](#)

- [How to render Angular Grid with material theme](#)
- [How to get started easily with Syncfusion angular 9 data grid?](#)
- [How to get started easily with Syncfusion angular 8 data grid?](#)
- [How to handle errors in Angular Grid component?](#)
- [How to get started easily with Syncfusion Angular 7 Data Grid/DataTable?](#)
- [How to get started easily with an example of Syncfusion angular 5 data grid?](#)
- [How to get started easily with Syncfusion angular 4 data grid?](#)

### Module in Angular Grid component

The following value providers should be injected to extend grid's functionality.

Module	Description
-----	-----
<a href="#">PageService</a>	Inject this module to use paging feature.
<a href="#">SortService</a>	Inject this module to use sorting feature.
<a href="#">FilterService</a>	Inject this module to use filtering feature.
<a href="#">GroupService</a>	Inject this module to use grouping feature.
<a href="#">EditService</a>	Inject this module to use editing feature.
<a href="#">AggregateService</a>	Inject this module to use aggregate feature.
<a href="#">ColumnChooserService</a>	Inject this module to use column chooser feature.
<a href="#">ColumnMenuService</a>	Inject this module to use column menu feature.
<a href="#">CommandColumnService</a>	Inject this module to use command column feature.
<a href="#">ContextMenuService</a>	Inject this module to use context menu feature.
<a href="#">DetailRowService</a>	Inject this module to use detail template feature.
<a href="#">ForeignKeyService</a>	Inject this module to use foreign key feature.
<a href="#">FreezeService</a>	Inject this module to use frozen rows and columns feature.
<a href="#">ResizeService</a>	Inject this module to use resize feature.
<a href="#">ReorderService</a>	Inject this module to use reorder feature.
<a href="#">RowDDService</a>	Inject this module to use row drag and drop feature.
<a href="#">SearchService</a>	Inject this module to use search feature and this is a default injected module.
<a href="#">SelectionService</a>	Inject this module to use selection feature and this is a default injected module.
<a href="#">ScrollService</a>	Inject this module to use scrolling feature and this is a default injected module.
<a href="#">PrintService</a>	Inject this module to use to use print feature and this is a default injected module.
<a href="#">ToolbarService</a>	Inject this module to use toolbar feature.
<a href="#">VirtualScrollService</a>	Inject this module to use virtual scroll feature.
<a href="#">ExcelExportService</a>	Inject this module to use excel export feature.

| PdfExportService | Inject this module to use PDF export feature. |

These modules should be injected into the **providers** section of root **NgModule** or component class.

### Data binding in Angular Grid component

Data binding is a fundamental technique that empowers the Grid component to integrate data into its interface, enabling the creation of dynamic and interactive grid views. This feature is particularly valuable when working with large datasets or when data needs to be fetched remotely.

The Syncfusion Grid utilizes the [Link to the Video](#), which supports both local binding with JavaScript object arrays and remote binding with RESTful JSON data services. The key property, [dataSource](#), can be assigned to a DataManager instance or a collection of JavaScript object arrays.

It supports two kinds of data binding methods:

- Local data
- Remote data

To learn about how to bind local, remote or observables data to Angular Grid, you can check on this video:

### Loading indicator

The Syncfusion Angular Grid offers a loading animation feature, which makes it easy to identify when data is being loaded or refreshed. This feature provides a clear understanding of the grid's current state and actions, such as sorting, filtering, grouping, and more.

To achieve this, you can utilize the [loadingIndicator.indicatorType](#) property of the grid, which supports two types of indicators:

- Spinner (default indicator)
- Shimmer

The following example demonstrates how to set the [loadingIndicator.indicatorType](#) property based on changing the dropdown value using the [change](#) event of the [DropDownList](#) component. The [refreshColumns](#) method is used to apply the changes and display the updated loading indicator type.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, SortService, FilterService } from
 '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { DropDownListComponent, ChangeEventArgs } from '@syncfusion/ej2-
angular-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
```

```

providers: [PageService, SortService, FilterService],
standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0"> Change the loading
indicator type: </label>
      <ejs-dropdownlist
        #dropdown
        style="margin-top: 5px"
        index="0"
        width="120"
        [dataSource]="ddlData"
        [fields]='fields'
        (change)="valueChange($event)">
      </ejs-dropdownlist>
    </div>
    <ejs-grid #grid id="grid" style="padding: 10px 10px" [dataSource]='data'
[allowPaging]='true' [allowSorting]='true' [allowFiltering]='true'
[pageSettings]='pageSettings' [loadingIndicator]='loadingIndicator'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
width='130' textAlign='Right'></e-column>
        <e-column field='Employees' headerText='Employee Name'
width='145'></e-column>
        <e-column field='Designation' headerText='Designation'
width='130'></e-column>
        <e-column field='CurrentSalary' headerText='Current Salary'
width='140' format="C2" textAlign='Right'></e-column>
      </e-columns>
    </ejs-grid>`
  })
export class AppComponent implements OnInit {
  public data?: DataManager;
  public loadingIndicator?: any;
  public pageSettings?: object;
  @ViewChild('grid')
  public grid?: GridComponent;
  @ViewChild('dropdown')
  public dropdown?: DropDownListComponent;
  public fields: object = { text: 'value', value: 'id' };
  public ddlData?: object[] = [
    { id: 'Spinner', value: 'Spinner' },
    { id: 'Shimmer', value: 'Shimmer' }
  ]
  ngOnInit(): void {
    this.data = new DataManager({ url:
'https://ej2services.syncfusion.com/js/development/api/UrlDataSource',
adaptor: new UrlAdaptor });
    this.loadingIndicator = { indicatorType: 'Spinner' };
    this.pageSettings = { pageCount: 3 };
  }
  valueChange(args: ChangeEventArgs) {
    if ((this.dropdown as DropDownListComponent).value === 'Shimmer') {
      (this.grid as GridComponent).loadingIndicator.indicatorType =
'Shimmer';
      (this.grid as GridComponent).refreshColumns();
    }
  }
}

```



```

    } else {
      (this.grid as GridComponent).loadingIndicator.indicatorType =
        'Spinner';
      (this.grid as GridComponent).refreshColumns();
    }
  }
}

```

### 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 datasource using property

Refreshing the data source in a Syncfusion Grid involves updating the data that the grid displays dynamically. This operation is essential when you need to reflect changes in the underlying data without reloading the entire page or component.

To achieve this, you can make use of the [datasource](#) property in conjunction with the [setProperties](#) method. This ensures that the grid reflects the changes in the data source without requiring a complete page or component reload.

For example, if you add or delete data source records, follow these steps:

**Step 1:** Add/delete the datasource record by using the following code.

```

`typescript
this.grid.dataSource.unshift(data); // Add a new record.
this.grid.dataSource.splice(selectedRow, 1); // Delete a record.
`

```

**Step 2:** Refresh the datasource after changes by invoking the `setProperties` method.

```

`typescript
(this.grid as GridComponent).setProperties({ dataSource: (this.grid as GridComponent).dataSource as
object[] });
`

```

The following example demonstrates adding a new record to the data source through an external button:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, PageService, ToolbarService } from
 '@syncfusion/ej2-angular-grids'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';

```

```

@Component({
  imports: [
    GridModule,
    ButtonAllModule
  ],
  providers: [EditService, PageService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="padding: 5px 0px 20px 0px ">
      <button ej-button id="sample" (click)='changeDatasource()'>
Refresh Datasource </button>
    </div>
    <ejs-grid #grid [dataSource]='data' [height]='280' >
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=120
></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  public newRecords:object ={};
  ngOnInit(): void {
    this.data = data;
  }
  changeDatasource(): void {
    for (let i = 0; i < 5; i++) {
      this.newRecords = {
        OrderID: this.generateOrderId(),
        CustomerID: this.generateCustomerId(),
        ShipCity: this.generateShipCity(),
        Freight: this.generateFreight(),
        ShipName: this.generateShipName()
      };
      ((this.grid as GridComponent).dataSource as object[]).unshift(this.newRecords);
      (this.grid as GridComponent).setProperties({ dataSource: (this.grid as GridComponent).dataSource as object[] });
    }
    // Generate a random OrderID
    generateOrderId(): number {
      return Math.floor(10000 + Math.random() * 90000);
    }
    // Generate a random CustomerID
    generateCustomerId(): string {
      const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';

```

```

    let result = '';
    for (let i = 0; i < 5; i++) {
        result += characters.charAt(Math.floor(Math.random() *
characters.length));
    }
    return result;
}
// Generate a random ShipCity
generateShipCity(): string {
    const cities = ['London', 'Paris', 'New York', 'Tokyo', 'Berlin'];
    return cities[Math.floor(Math.random() * cities.length)];
}
// Generate a random Freight value
generateFreight(): number {
    return Math.random() * 100;
}
// Generate a random ShipName
generateShipName(): string {
    const names = ['Que Delícia', 'Bueno Foods', 'Island Trading',
'Laughing Bacchus Winecellars'];
    return names[Math.floor(Math.random() * names.length)];
}
}

```

## MAIN.TS

```

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

```

### Dynamically change the datasource or columns or both

The Grid component in Syncfusion allows dynamic modification of the data source, columns, or both . This feature is particularly valuable when you need to refresh the grid's content and structure without requiring a complete page reload.

To achieve dynamic changes, you can utilize the [changeDataSource](#) method. This method enables you to update the data source, columns, or both, based on your application's requirements. However, it is important to note that during the changing process for the data source and columns, the grid's existing actions such as sorting, filtering, grouping, aggregation, and searching will be reset. The `changeDataSource` method has two optional arguments: the first argument represents the data source, and the second argument represents the columns. The various uses of the `changeDataSource` method are explained in the following topic.

#### 1. Change both data source and columns:

To modify both the existing columns and the data source, you need to pass the both arguments to the `changeDataSource` method. The following example demonstrates how to change both the data source and columns.

You can assign a JavaScript object array to the [dataSource](#) property to bind local data to the grid. The code below provides an example of how to create a data source for the grid.

`typescript

```
export let data: Object[] = [
{
  OrderID: 10248, CustomerID: 'VINET', Freight: 32.38,
  ShipCity: 'Reims'
},
{
  OrderID: 10249, CustomerID: 'TOMSP', Freight: 11.61,
  ShipCity: 'Münster'
},
{
  OrderID: 10250, CustomerID: 'HANAR', Freight: 61.34,
  ShipCity: 'Rio de Janeiro'
}];
`
```

The following code demonstrates how to create the [columns](#) for the grid, which are based on the provided grid data source.

```
`typescript
public newColumn: ColumnModel[] = [
{ field: 'OrderID', headerText: 'Order ID', textAlign: 'Right', width: 125 },
{ field: 'CustomerID', headerText: 'Customer ID', width: 125 },
];
`
```

The following code demonstrates updating the data source and columns defined above using the `changeDataSource` method.

```
`typescript
this.gridInstance.changeDataSource(data, newColumn);
`
```

## 2. Modify only the existing columns:

To modify the existing columns in a grid, you can either add or remove columns or change the entire set of columns using the [changeDataSource](#) method. To use this method, you should set the first parameter to null and provide the new columns as the second parameter. However, please note that if a column field is not specified in the data source, its corresponding column values will be empty. The following example illustrates how to modify existing columns.

The following code demonstrates how to add new columns with existing grid columns ('newColumn') by using `changeDataSource` method.

```
`typescript
public newColumn1: ColumnModel[] = [
  { field: 'Freight', headerText: 'Freight', textAlign: 'Right', width: 125 },
  { field: 'ShipCity', headerText: 'ShipCity', width: 125 },
];
let column: any = this.newColumn.push(...this.newColumn1);
this.gridInstance.changeDataSource(null, column);
`
```

### 3. Modify only the data source:

You can change the entire data source in the grid using the `changeDataSource` method. To use this method, you should provide the data source as the first argument, and the second argument which is optional can be used to specify new columns for the grid. If you are not specifying the columns, the grid will generate the columns automatically based on the data source. The following example demonstrates how to modify the data source.

You can assign a JavaScript object array to the `dataSource` property to bind local data to the grid. The code below provides an example of how to create a new data source for the grid.

```
`typescript
export let employeeData: Object[] = [
  {
    FirstName: 'Nancy', City: 'Seattle', Region: 'WA',
    Country: 'USA'
  },
  {
    FirstName: 'Andrew', City: 'London', Region: null,
    Country: 'UK',
  },
  {
    FirstName: 'Janet', City: 'Kirkland', Region: 'WA',
    Country: 'USA'
  }
];
`
```

The following code demonstrates, how to use the `changeDataSource` method to bind the new **employeeData** to the grid.

```
`typescript
this.gridInstance.changeDataSource(employeeData);
`
```

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { GridModule, PageService, SortService, FilterService, GroupService }
from '@syncfusion/ej2-angular-grids'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import {data} from './datasource';
@Component({
  imports: [

    FormsModule,
    GridModule,
    ButtonAllModule
  ],
  providers: [
    PageService,
    SortService,
    FilterService,
    GroupService
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div style="padding:0px 0px 20px 0px">
    <button ej2-button (click)=next($event)>Change datasource
and column</button>
    </div>
    <ejs-grid #Grid [dataSource]="data" allowPaging="true"
[pageSettings]="pageSettings" >
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
textAlign='Right' width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' width=120></e-column>
    </e-columns>
    </ejs-grid>`
  })
export class AppComponent implements OnInit {
  public data?: object[];
  public isDataLoading = true;
  public pageSettings?: object = { pageSize: 5, pageCount: 3 }
  @ViewChild('Grid') public grid?: GridComponent;
  public count = 0;
  public newColumn = [
    { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 125 },
    { field: 'CustomerName', headerText: 'Customer Name', width: 125 },
    { field: 'OrderDate', headerText: 'Order Date', width: 130, format:
'yMd', textAlign: 'Right'},

```

```

        { field: 'Freight', headerText: 'Freight', width: 120, textAlign:
'Right'},
    ];
    ngOnInit(): void {
        this.data= data

    }
    next(args:MouseEvent) {
        this.count = this.count + 100;
        (this.grid as GridComponent).changeDataSource(data.slice(0, this.count
+ 100), this.newColumn as Object[]);
    }
}

```

### MAIN.TS

```

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

```

\* The Grid state persistence feature does not support the `changeDataSource` method.

\* In this document, the above sample uses the local data for `changeDataSource` method. For those using a remote data source, refer to the [FlexibleData](#) resource.

See also

- [Binding a firebase data source to Grid using AngularFire2](#)
- [How to bind SQL Server data in Angular DataGrid using SqlClient data provider](#)

## Connecting to Adaptors

### UrlAdaptor in Syncfusion Angular Grid Component

The [UrlAdaptor](#) serves as the base adaptor for facilitating communication between remote data services and an UI component. It enables seamless data binding and interaction with custom API services or any remote service through URLs. The UrlAdaptor is particularly useful for the scenarios where a custom API service with unique logic for handling data and CRUD operations is in place. This approach allows for custom handling of data and CRUD operations, and the resultant data returned in the `result` and `count` format for display in the Syncfusion Angular Grid component.

This section describes a step-by-step process for retrieving data using UrlAdaptor, then binding it to the Angular Grid component to facilitate data and CRUD operations.

#### Creating an API service

To configure a server with Syncfusion Angular Grid, you need to follow the below steps:

#### 1. Project Creation:

Open Visual Studio and create an Angular and ASP.NET Core project named **UrlAdaptor**. To create an Angular and ASP.NET Core application, follow the documentation [link](#) for detailed steps.

#### 2. Model Class Creation:

Create a model class named **OrdersDetails.cs** in the server-side **Models** folder to represent the order data.

#### **ORDERSDETAILS.CS**

```
namespace UrlAdaptor.Server.Models
{
    public class OrdersDetails
    {
        public static List<OrdersDetails> order = new List<OrdersDetails>();
        public OrdersDetails()
        {
        }
        public OrdersDetails(
            int OrderID, string CustomerId, int EmployeeId, double Freight, bool
            Verified,
            DateTime OrderDate, string ShipCity, string ShipName, string ShipCountry,
            DateTime ShippedDate, string ShipAddress)
        {
            this.OrderID = OrderID;
            this.CustomerID = CustomerId;
            this.EmployeeID = EmployeeId;
            this.Freight = Freight;
            this.ShipCity = ShipCity;
            this.Verified = Verified;
            this.OrderDate = OrderDate;
            this.ShipName = ShipName;
            this.ShipCountry = ShipCountry;
            this.ShippedDate = ShippedDate;
            this.ShipAddress = ShipAddress;
        }
        public static List<OrdersDetails> GetAllRecords()
        {
            if (order.Count() == 0)
            {
                int code = 10000;
                for (int i = 1; i < 10; i++)
                {
                    order.Add(new OrdersDetails(code + 1, "ALFKI", i + 0, 2.3 * i, false, new
                    DateTime(1991, 05, 15), "Berlin", "Simons bistro", "Denmark", new
                    DateTime(1996, 7, 16), "Kirchgasse 6"));
                    order.Add(new OrdersDetails(code + 2, "ANATR", i + 2, 3.3 * i, true, new
                    DateTime(1990, 04, 04), "Madrid", "Queen Cozinha", "Brazil", new
                    DateTime(1996, 9, 11), "Avda. Azteca 123"));
                    order.Add(new OrdersDetails(code + 3, "ANTON", i + 1, 4.3 * i, true, new
                    DateTime(1957, 11, 30), "Cholchester", "Frankenversand", "Germany", new
                    DateTime(1996, 10, 7), "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
                    order.Add(new OrdersDetails(code + 4, "BLONP", i + 3, 5.3 * i, false, new
                    DateTime(1930, 10, 22), "Marseille", "Ernst Handel", "Austria", new
                    DateTime(1996, 12, 30), "Magazinweg 7"));
                    order.Add(new OrdersDetails(code + 5, "BOLID", i + 4, 6.3 * i, true, new
                    DateTime(1953, 02, 18), "Tsawassen", "Hanari Carnes", "Switzerland", new
                    DateTime(1997, 12, 3), "1029 - 12th Ave. S.));
                    code += 5;
                }
            }
            return order;
        }
    }
}
```



```

}
public int? OrderID { get; set; }
public string? CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string? ShipCity { get; set; }
public bool? Verified { get; set; }
public DateTime OrderDate { get; set; }
public string? ShipName { get; set; }
public string? ShipCountry { get; set; }
public DateTime ShippedDate { get; set; }
public string? ShipAddress { get; set; }
}
}

```

### 3. API Controller Creation:

Create a file named `GridController.cs` under the **Controllers** folder. This controller will handle data communication with the Angular Grid component.

#### GRIDCONTROLLER.CS

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using UrlAdaptor.Server.Models;
namespace UrlAdaptor.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class GridController : ControllerBase
    {
        [HttpPost]
        public object Post()
        {
            // Retrieve data from the data source (e.g., database)
            IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
            // Get the total records count
            int totalRecordsCount = DataSource.Count();
            // Return data based on the request
            return new { result = DataSource, count = totalRecordsCount };
        }
        [HttpGet]
        public List<OrdersDetails> GetOrderData()
        {
            var data = OrdersDetails.GetAllRecords().ToList();
            return data;
        }
    }
}

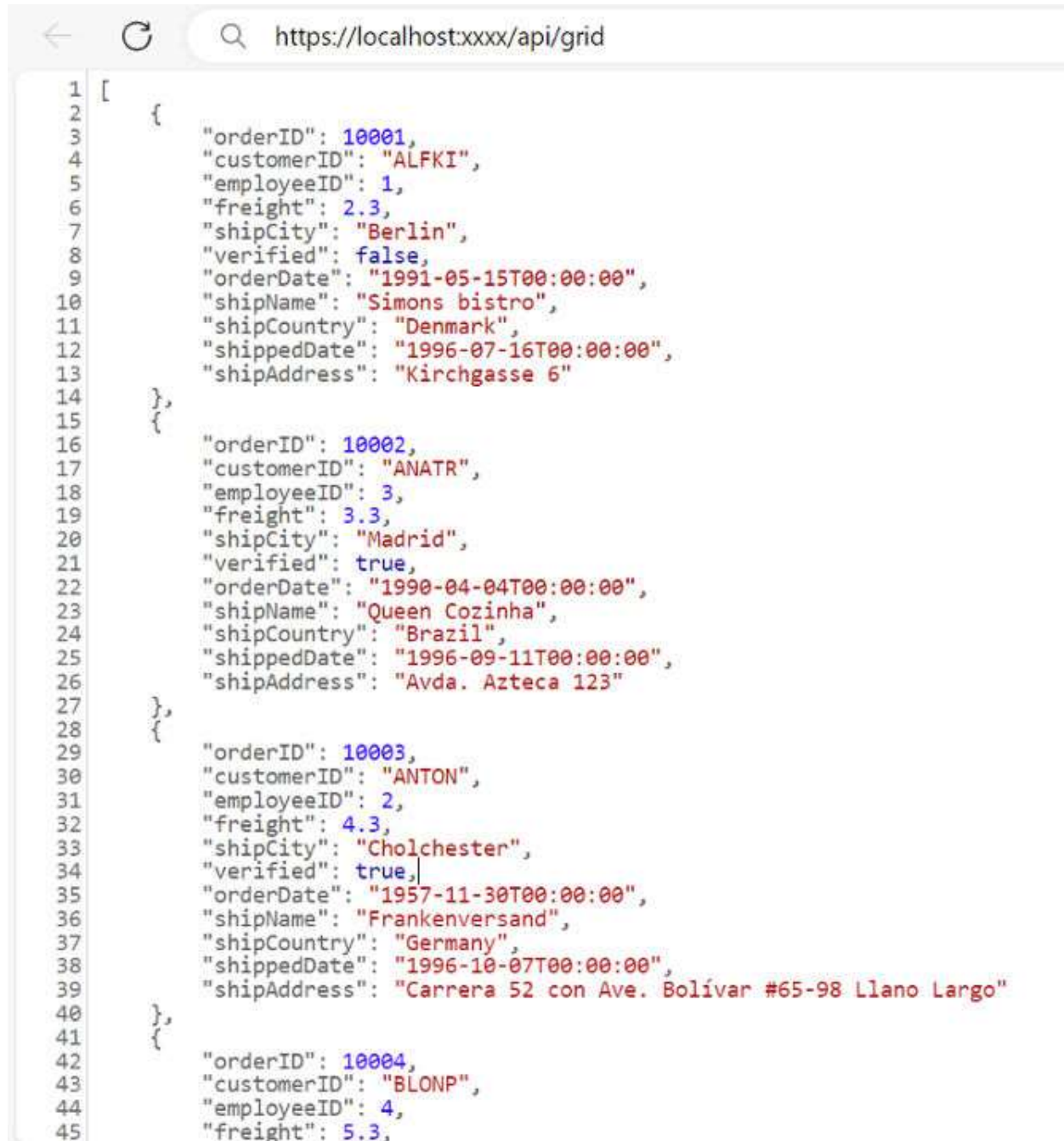
```

The **GetOrderData** method retrieves sample order data. You can replace it with your custom logic to fetch data from a database or any other source.

### 4. Run the Application:

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

After running the application, you can verify that the server-side API controller is successfully returning the order data in the URL(<https://localhost:xxxx/api/Grid>). Here **xxxx** denotes the port number.



```

1  [
2    {
3      "orderId": 10001,
4      "customerID": "ALFKI",
5      "employeeID": 1,
6      "freight": 2.3,
7      "shipCity": "Berlin",
8      "verified": false,
9      "orderDate": "1991-05-15T00:00:00",
10     "shipName": "Simons bistro",
11     "shipCountry": "Denmark",
12     "shippedDate": "1996-07-16T00:00:00",
13     "shipAddress": "Kirchgasse 6"
14   },
15   {
16     "orderId": 10002,
17     "customerID": "ANATR",
18     "employeeID": 3,
19     "freight": 3.3,
20     "shipCity": "Madrid",
21     "verified": true,
22     "orderDate": "1990-04-04T00:00:00",
23     "shipName": "Queen Cozinha",
24     "shipCountry": "Brazil",
25     "shippedDate": "1996-09-11T00:00:00",
26     "shipAddress": "Avda. Azteca 123"
27   },
28   {
29     "orderId": 10003,
30     "customerID": "ANTON",
31     "employeeID": 2,
32     "freight": 4.3,
33     "shipCity": "Cholchester",
34     "verified": true,
35     "orderDate": "1957-11-30T00:00:00",
36     "shipName": "Frankenversand",
37     "shipCountry": "Germany",
38     "shippedDate": "1996-10-07T00:00:00",
39     "shipAddress": "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"
40   },
41   {
42     "orderId": 10004,
43     "customerID": "BLONP",
44     "employeeID": 4,
45     "freight": 5.3,
  
```

#### *Connecting Syncfusion Angular Grid to an API service*

To integrate the Syncfusion Grid component into your Angular and ASP.NET Core project using Visual Studio, follow the below steps:

#### **1: Install Syncfusion Package**

Open your terminal in the project's client folder and install the required Syncfusion packages using npm:

```
`bash
```

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

```
npm install @syncfusion/ej2-data --save
```

## 2: Import Grid Module

In the `app.module.ts` file, import the **GridModule** from the `@syncfusion/ej2-angular-grids` package:

### Step 3: Adding CSS reference

Include the necessary CSS files in your `styles.css` file to style the Syncfusion Angular component:

#### 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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

### Step 4: Adding Syncfusion Component

In your component file (e.g., `app.component.ts`), import **DataManager** and **UrlAdaptor** from `@syncfusion/ej2-data`. Create a **DataManager** instance specifying the URL of your API endpoint(`https://localhost:xxxx/api/Grid`) using the `url` property and set the adaptor **UrlAdaptor**.

#### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/grid', // Here xxxx represents the port
      number
    });
    adaptor: new UrlAdaptor()
  });
}
```

#### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data'>
<e-columns>
```

```
<e-column field='OrderID' headerText='Order ID' width='120'  
textAlign='Right' isPrimaryKey="true"></e-column>  
<e-column field='CustomerID' headerText='Customer ID' width='160'></e-  
column>  
<e-column field='ShipCity' headerText='Ship City' width='150'></e-column>  
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-  
column>  
</e-columns>  
</ejs-grid>
```

Replace `https://localhost:/api/grid` with the actual **URL** of your API endpoint that provides the data in a consumable format (e.g., JSON).

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

Ensure your API service is configured to handle CORS (Cross-Origin Resource Sharing) if necessary.

```
`cs  
[program.cs]  
builder.Services.AddCors(options =>  
{  
options.AddDefaultPolicy(builder =>  
{  
builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();  
});  
});  
var app = builder.Build();  
app.UseCors();  
`
```

You can find the complete sample of `UrlAdaptor` in the [GitHub](#) link.

Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BLONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BLONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil
10013	ANTON	Cholchester	Germany

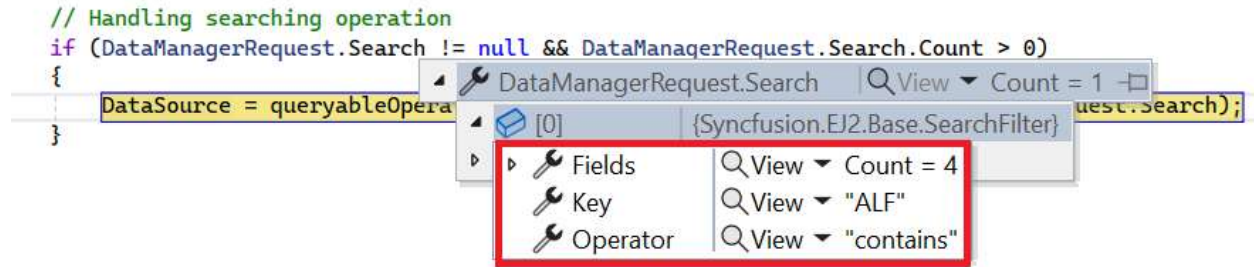
\* The Syncfusion Grid component provides built-in support for handling various data operations such as searching, sorting, filtering, aggregate and paging on the server-side. These operations can be handled using methods such as [PerformSearching](#), [PerformFiltering](#), [PerformSorting](#), [PerformTake](#) and [PerformSkip](#) available in the `Syncfusion.EJ2.AspNet.Core` package. Let's explore how to manage these data operations using the `UrlAdaptor`.

\* In an API service project, add `Syncfusion.EJ2.AspNet.Core` by opening the NuGet package manager in Visual Studio (Tools → NuGet Package Manager → Manage NuGet Packages for Solution), search and install it.

\* To access [DataManagerRequest](#) and [QueryableOperation](#), import `Syncfusion.EJ2.Base` in `GridController.cs` file.

#### *Handling searching operation*

To handle searching operation, ensure that your API endpoint supports custom searching criteria. Implement the searching logic on the server-side using the [PerformSearching](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo searching based on the criteria specified in the incoming [DataManagerRequest](#) object.



### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManagerRequest DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Handling Searching
    if (DataManagerRequest.Search != null && DataManagerRequest.Search.Count >
0)
    {
        DataSource = queryableOperation.PerformSearching(DataSource,
DataManagerRequest.Search);
    }
    // Get the total records count
    int totalRecordsCount = DataSource.Count();
    // Return data based on the request
    return new { result = DataSource, count = totalRecordsCount };
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
})
export class AppComponent {
    public data?: DataManager;
    @ViewChild('grid')
    public grid?: GridComponent;
    public toolbar?: ToolbarItems[];
    ngOnInit(): void {
        this.data = new DataManager({
            url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
            adaptor: new UrlAdaptor()
        });
        this.toolbar = ['Search'];
    }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [toolbar]="toolbar" height="320">
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='ShipCity' headerText='ShipCity' width='150'
textAlign='Right'></e-column>
</e-columns>
</ejs-grid>
```

### Handling filtering operation

To handle filtering operation, ensure that your API endpoint supports custom filtering criteria. Implement the filtering logic on the server-side using the [PerformFiltering](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo filtering based on the criteria specified in the incoming [DataManagerRequest](#) object.

#### Single column filtering

The screenshot shows a C# code snippet for handling filtering operations. The code iterates through conditions and predicates in a `DataManagerRequest` object. The debugger window on the right shows the state of `DataManagerRequest.Where` with the following properties:

Property	Value
Condition	null
Field	"CustomerID"
IgnoreCase	true
IsComplex	false
Operator	"startswith"
predicates	null
value	ValueKind = String : "ANT"

#### Multi column filtering

The screenshot shows a C# code snippet for handling multi-column filtering. The debugger window on the right shows the state of `DataManagerRequest.Where` with the following properties:

Property	Value
Condition	"and"
Field	null
IgnoreCase	true
IsComplex	true
Operator	null
predicates	Count = 2
[0]	(Syncfusion.EJ2.Base.WhereFilter)
[1]	(Syncfusion.EJ2.Base.WhereFilter)

#### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManagerRequest DataManagerRequest)
{
}
```



```
// Retrieve data from the data source (e.g., database)
IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
QueryableOperation queryableOperation = new QueryableOperation(); //
Initialize QueryableOperation instance
if (DataManagerRequest.Where != null && DataManagerRequest.Where.Count > 0)
{
    // Handling filtering operation
    foreach (var condition in DataManagerRequest.Where)
    {
        foreach (var predicate in condition.predicates)
        {
            DataSource = queryableOperation.PerformFiltering(DataSource,
            DataManagerRequest.Where, predicate.Operator);
        }
    }
}
// Get the total records count
int totalRecordsCount = DataSource.Count();
// Return data based on the request
return new { result = DataSource, count = totalRecordsCount };
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
})
export class AppComponent {
    public data?: DataManager;
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = new DataManager({
            url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
            adaptor: new UrlAdaptor()
        });
    }
}
```

### APP.COMPONENT.HTML

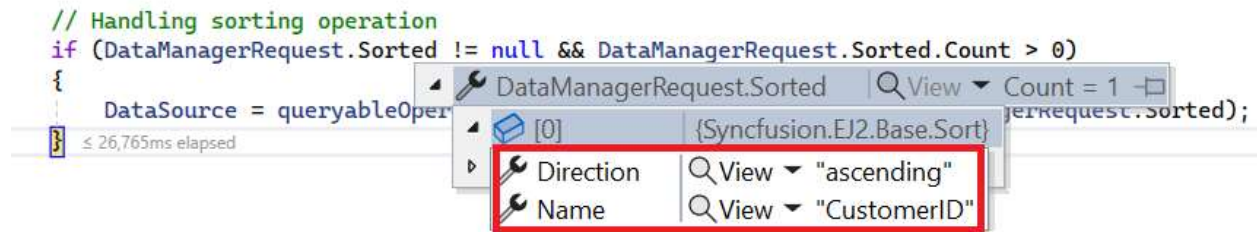
```
<ejs-grid #grid [dataSource]='data' allowFiltering='true' height="320">
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='ShipCity' headerText='ShipCity' width='150'
textAlign='Right'></e-column>
</e-columns>
</ejs-grid>
```



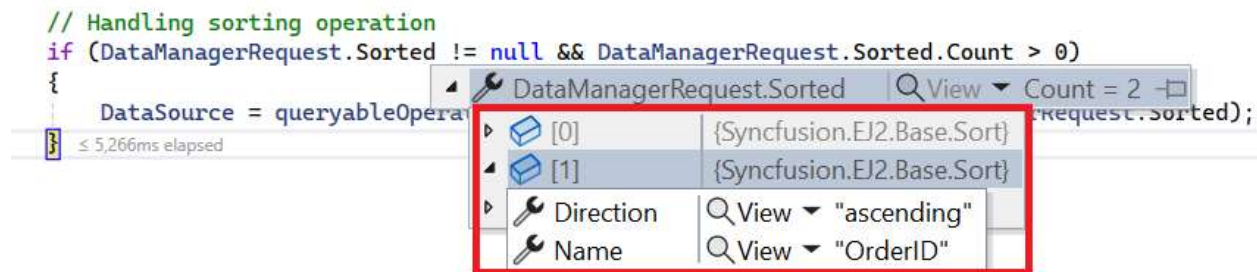
### Handling sorting operation

To handle sorting operation, ensure that your API endpoint supports custom sorting criteria. Implement the sorting logic on the server-side using the [PerformSorting](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo sorting based on the criteria specified in the incoming [DataManagerRequest](#) object.

### Single column sorting



### Multi column sorting



### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManagerRequest DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Handling Sorting operation
    if (DataManagerRequest.Sorted != null && DataManagerRequest.Sorted.Count > 0)
    {
        DataSource = queryableOperation.PerformSorting(DataSource,
        DataManagerRequest.Sorted);
    }
    // Get the total count of records
    int totalRecordsCount = DataSource.Count();
    // Return data based on the request
    return new { result = DataSource, count = totalRecordsCount };
}
```

### APP.COMPONENT.TS

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

```
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
      adaptor: new UrlAdaptor()
    });
  }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' allowSorting='true' height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
      column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
  </e-columns>
</ejs-grid>
```

### Handling paging operation

To handle paging operation, ensure that your API endpoint supports custom paging criteria. Implement the paging logic on the server-side using the [PerformTake](#) and [PerformSkip](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo paging based on the criteria specified in the incoming [DataManagerRequest](#) object.

```
public object Post([FromBody] DataManagerRequest DataManagerRequest)
{
  // Retrieve data from the data source (e.g., database)
  IQueryable<OrdersDetails> DataSource = GetOrderData();

  QueryableOperation queryableOperation = new QueryableOperation(DataSource);

  // Handling paging operation.
  if (DataManagerRequest.Skip != 0)
  {
    DataSource = queryableOperation.PerformSkip(DataManagerRequest.Skip);
  }
  if (DataManagerRequest.Take != 0)
  {
    DataSource = queryableOperation.PerformTake(DataManagerRequest.Take);
  }

  return DataSource.ToList();
}
```

DataManagerRequest (Syncfusion.EJ2.Base.DataManagerRequest)	
Aggregates	null
Expand	null
Group	null
IsLazyLoad	false
OnDemandGroupInfo	null
RequiresCounts	true
Search	null
Select	null
<b>Skip</b>	<b>12</b>
Sorted	null
Table	null
<b>Take</b>	<b>12</b>
Where	null
antiForgery	null

### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManagerRequest DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    // Get the total records count
    int totalRecordsCount = DataSource.Count();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Handling paging operation.
    if (DataManagerRequest.Skip != 0)
    {
        // Paging
        DataSource = queryableOperation.PerformSkip(DataSource,
        DataManagerRequest.Skip);
    }
    if (DataManagerRequest.Take != 0)
    {
        DataSource = queryableOperation.PerformTake(DataSource,
        DataManagerRequest.Take);
    }
    // Return data based on the request
    return new { result = DataSource, count = totalRecordsCount };
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
})
export class AppComponent {
    public data?: DataManager;
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = new DataManager({
            url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
            adaptor: new UrlAdaptor()
        });
    }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' allowPaging='true' height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
    width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
    column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
    textAlign='Right'></e-column>
```

```
</e-columns>
</ejs-grid>
```

### Handling CRUD operations

The Syncfusion Angular Grid Component seamlessly integrates CRUD (Create, Read, Update, Delete) operations with server-side controller actions through specific properties: [insertUrl](#), [removeUrl](#), [updateUrl](#), [crudUrl](#), and [batchUrl](#). These properties enable the grid to communicate with the data service for every grid action, facilitating server-side operations.

### CRUD Operations Mapping

CRUD operations within the grid can be mapped to server-side controller actions using specific properties:

1. **insertUrl**: Specifies the URL for inserting new data.
2. **removeUrl**: Specifies the URL for removing existing data.
3. **updateUrl**: Specifies the URL for updating existing data.
4. **crudUrl**: Specifies a single URL for all CRUD operations.
5. **batchUrl**: Specifies the URL for batch editing.

To enable editing in Angular Grid component, refer to the editing [documentation](#). In the below example, the inline edit [mode](#) is enabled and [toolbar](#) property is configured to display toolbar items for editing purposes.

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
      insertUrl: 'https://localhost:xxxx/api/grid/Insert',
      updateUrl: 'https://localhost:xxxx/api/grid/Update',
      removeUrl: 'https://localhost:xxxx/api/grid/Remove',
      adaptor: new UrlAdaptor()
    });
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel'];
    this.editSettings = { allowAdding: true, allowDeleting: true, allowEditing: true, mode: 'Batch' };
  }
}
```

**APP.COMPONENT.HTML**

```

<ejs-grid #grid [dataSource]='data' [toolbar]="toolbar"
[editSettings]="editSettings">
<e-columns>
<e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right' isPrimaryKey="true"></e-column>
<e-column field='CustomerID' headerText='Customer ID' width='160'></e-
column>
<e-column field='ShipCity' headerText='Ship City' width='150'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>

```

Normal/Inline editing is the default edit [mode](#) for the Grid component. To enable CRUD operations, ensure that the [isPrimaryKey](#) property is set to **true** for a specific Grid column, ensuring that its value is unique.

The below class is used to structure data sent during CRUD operations.

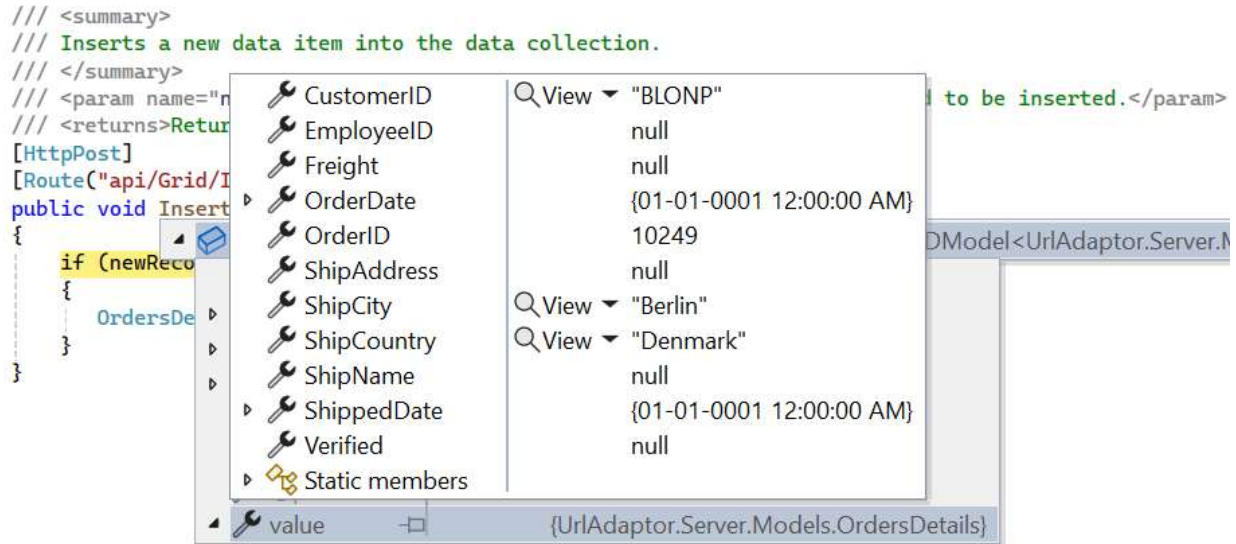
```

`cs
public class CRUDModel<T> where T : class
{
public string? action { get; set; }
public string? keyColumn { get; set; }
public object? key { get; set; }
public T? value { get; set; }
public List<T>? added { get; set; }
public List<T>? changed { get; set; }
public List<T>? deleted { get; set; }
public IDictionary<string, object>? @params { get; set; }
}
`

```

**Insert operation:**

To insert a new record, utilize the [insertUrl](#) property to specify the controller action mapping URL for the insert operation. The newly added record details are bound to the **newRecord** parameter.

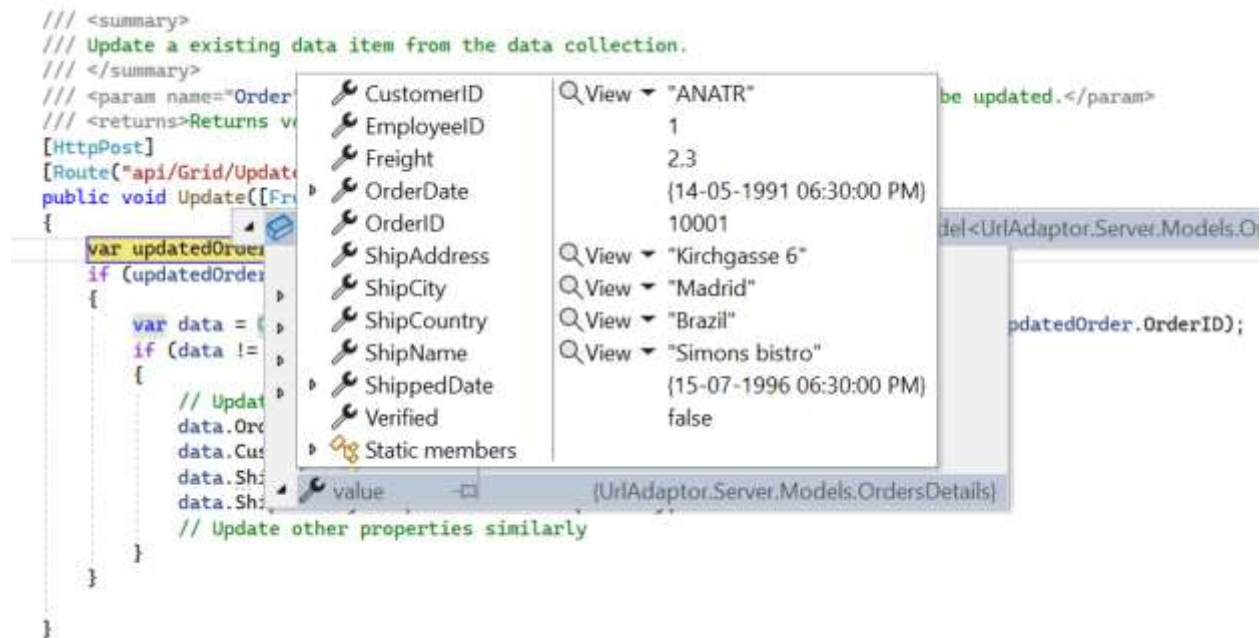


```
`cs
```

```
/// <summary>
/// Inserts a new data item into the data collection.
/// </summary>
/// <param name="newRecord">It contains the new record detail which is need to be
inserted.</param>
/// <returns>Returns void</returns>
[HttpPost]
[Route("api/Grid/Insert")]
public void Insert([FromBody] CRUDModel<OrdersDetails> newRecord)
{
    if (newRecord.value != null)
    {
        OrdersDetails.GetAllRecords().Insert(0, newRecord.value);
    }
}
`
```

### Update operation:

For updating existing records, utilize the [updateUrl](#) property to specify the controller action mapping URL for the update operation. The updated record details are bound to the **updatedRecord** parameter.



```
`cs
```

```

/// <summary>
/// Update a existing data item from the data collection.
/// </summary>
/// <param name="updatedRecord">It contains the updated record detail which is need to be
updated.</param>
/// <returns>Returns void</returns>
[HttpPost]
[Route("api/Grid/Update")]
public void Update([FromBody] CRUDModel<OrdersDetails> updatedRecord)
{
    var updatedOrder = updatedRecord.value;
    if (updatedOrder != null)
    {
        var data = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID == updatedOrder.OrderID);
        if (data != null)
        {
            // Update the existing record
            data.OrderID = updatedOrder.OrderID;
            data.CustomerID = updatedOrder.CustomerID;
            data.ShipCity = updatedOrder.ShipCity;

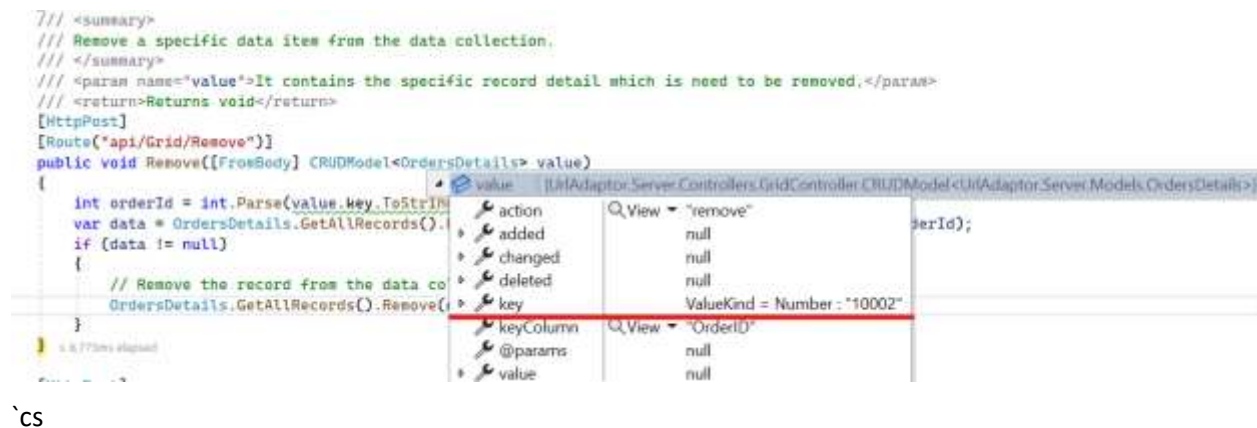
```



```
data.ShipCountry = updatedOrder.ShipCountry;
// Update other properties similarly
}
}
}
,
```

### Delete operation

To delete existing records, use the [removeUrl](#) property to specify the controller action mapping URL for the delete operation. The primary key value of the deleted record is bound to the **deletedRecord** parameter.



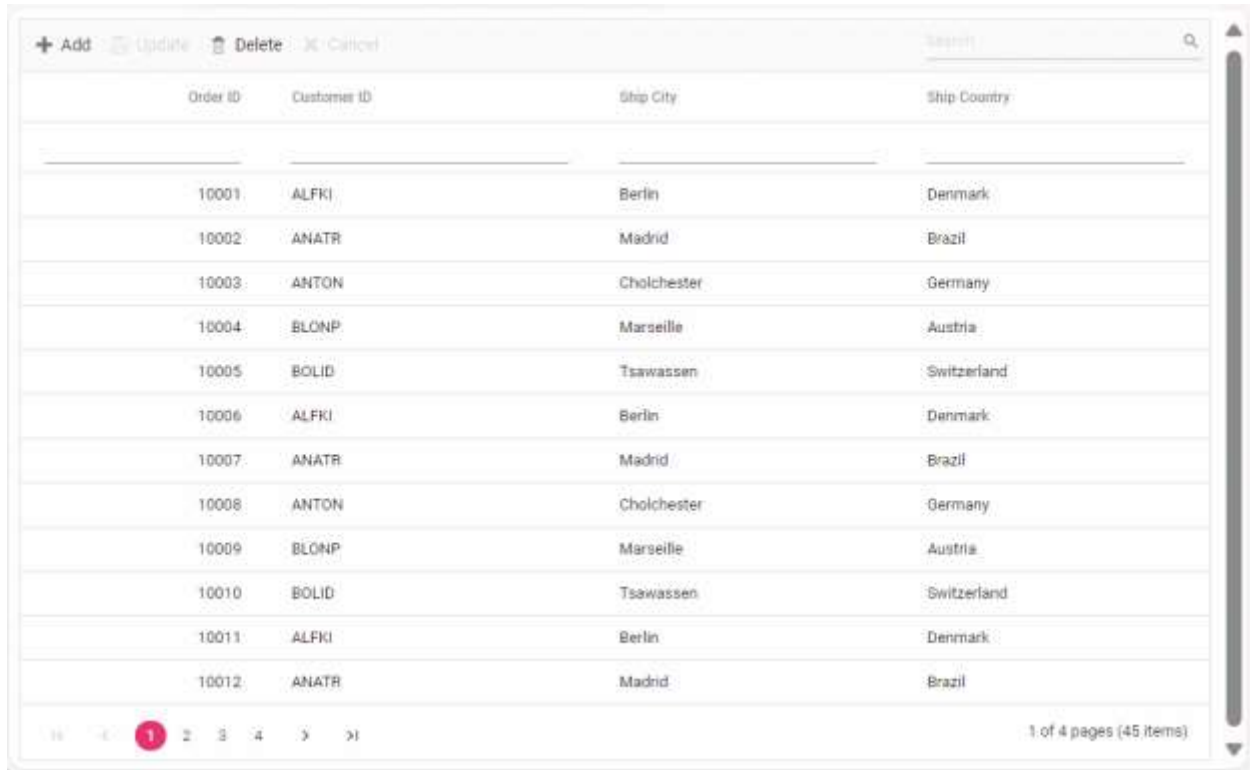
```
`cs
/// <summary>
/// Remove a specific data item from the data collection.
/// </summary>
/// <param name="deletedRecord">It contains the specific record detail which is need to be
removed.</param>
/// <return>Returns void</return>
[HttpPost]
[Route("api/Grid/Remove")]
public void Remove([FromBody] CRUDModel<OrdersDetails> deletedRecord)
{
    int orderId = int.Parse(deletedRecord.key.ToString()); // get key value from the deletedRecord
    var data = OrdersDetails.GetAllRecords().FirstOrDefault(orderData => orderData.OrderID == orderId);
    if (data != null)
    {
        // Remove the record from the data collection
        OrdersDetails.GetAllRecords().Remove(data);
    }
}
```



```

}
}
,

```



Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BLONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BLONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil

You can find the complete sample for the UrlAdaptor in [GitHub](#) link.

### Single method for performing all CRUD operations

Using the `crudUrl` property, the controller action mapping URL can be specified to perform all the CRUD operation at server-side using a single method instead of specifying separate controller action method for CRUD (insert, update and delete) operations.

The following code example describes the above behavior.

```

`ts
import { Component, ViewChild } from '@angular/core';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {

```

```

public data?: DataManager;
@ViewChild('grid')
public grid?: GridComponent;
public editSettings?: EditSettingsModel;
public toolbar?: ToolbarItems[];
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
crudUrl: 'https://localhost:xxxx/api/grid/CrudUpdate',
adaptor: new UrlAdaptor()
});
this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel'];
this.editSettings = { allowAdding: true, allowDeleting: true, allowEditing: true, mode: 'Normal' };
}
}
`
`cs
[HttpPost]
[Route("api/[controller]/CrudUpdate")]
public void CrudUpdate([FromBody] CRUDModel<OrdersDetails> request)
{
// perform update operation
if (request.action == "update")
{
var orderValue = request.value;
OrdersDetails existingRecord = OrdersDetails.GetAllRecords().Where(or => or.OrderID ==
orderValue.OrderID).FirstOrDefault();
existingRecord.OrderID = orderValue.OrderID;
existingRecord.CustomerID = orderValue.CustomerID;
existingRecord.ShipCity = orderValue.ShipCity;
}
// perform insert operation
else if (request.action == "insert")

```

```

{
OrdersDetails.GetAllRecords().Insert(0, request.value);
}
// perform remove operation
else if (request.action == "remove")
{
OrdersDetails.GetAllRecords().Remove(OrdersDetails.GetAllRecords().Where(or => or.OrderID ==
int.Parse(request.key.ToString()))).FirstOrDefault());
}
}
,

```

### Batch operation

To perform batch operation, define the edit [mode](#) as **Batch** and specify the `batchUrl` property in the `DataManager`. Use the **Add** toolbar button to insert new row in batch editing mode. To edit a cell, double-click the desired cell and update the value as required. To delete a record, simply select the record and press the **Delete** toolbar button. Now, all CRUD operations will be executed in single request. Clicking the **Update** toolbar button will update the newly added, edited, or deleted records from the `OrdersDetails` table using a single API POST request.

```

`ts
[app.component.ts]
import { Component, ViewChild } from '@angular/core';
import { GridComponent, ToolbarItems, EditSettingsModel } from '@syncfusion/ej2-angular-grids';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
@Component({
selector: 'app-root',
template: `<ejs-grid #grid [dataSource]='data' [editSettings]='editSettings' [toolbar]='toolbar'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' width='90' textAlign='Right' isPrimaryKey='true'></e-
column>
<e-column field="CustomerID" headerText="Customer Name" width="100"></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=100></e-column>
</e-columns>
</ejs-grid>`
})
export class AppComponent {

```

```

@ViewChild('grid')
public grid?: GridComponent;
public data?: DataManager;
public editSettings?: EditSettingsModel;
public toolbar?: ToolbarItems[];
public orderIDRules?: object;
public customerIDRules?: object;
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
batchUrl: 'https://localhost:xxxx/api/grid/BatchUpdate',
adaptor: new UrlAdaptor()
});
this.editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Batch' };
this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'];
this.orderIDRules = { required: true };
this.customerIDRules = { required: true, minLength: 3 };
}
}
`cs
public IActionResult BatchUpdate([FromBody] CRUDModel<OrdersDetails> batchOperation)
{
if (batchOperation.added != null)
{
foreach (var addedOrder in batchOperation.added)
{
OrdersDetails.GetAllRecords().Insert(0, addedOrder);
}
}
if (batchOperation.changed != null)
{
foreach (var changedOrder in batchOperation.changed)

```

```
{
var existingOrder = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID ==
changedOrder.OrderID);
if (existingOrder != null)
{
existingOrder.CustomerID = changedOrder.CustomerID;
existingOrder.ShipCity = changedOrder.ShipCity;
// Update other properties as needed
}
}
}
if (batchOperation.deleted != null)
{
foreach (var deletedOrder in batchOperation.deleted)
{
var orderToDelete = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID ==
deletedOrder.OrderID);
if (orderToDelete != null)
{
OrdersDetails.GetAllRecords().Remove(orderToDelete);
}
}
}
return Json(batchOperation);
}
、
```

<div> <div>+ Add</div> <div>Update</div> <div>Delete</div> <div>Cancel</div> </div> <div>Search</div>			
Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BLONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BLONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil
<div> <div>&lt;&lt;</div> <div>&lt;</div> <div>1</div> <div>2</div> <div>3</div> <div>4</div> <div>&gt;</div> <div>&gt;&gt;</div> </div> <div>1 of 4 pages (45 items)</div>			

### ODataV4Adaptor in Syncfusion Angular Grid Component

The **ODataV4Adaptor** in the Syncfusion Angular Grid Component allows seamless integration of the Angular Grid with OData v4 services, enabling efficient data fetching and manipulation. This guide provides detailed instructions on binding data and performing CRUD (Create, Read, Update, Delete) actions using the **ODataV4Adaptor** in your Syncfusion Angular Grid Component.

#### Creating an OData service

To configure a server with Syncfusion Angular Grid, you need to follow the below steps:

#### 1. Project Creation:

Open Visual Studio and create an Angular and ASP.NET Core project named **ODataV4Adaptor**. To create an Angular and ASP.NET Core application, follow the documentation [link](#) for detailed steps.

#### 2. Install NuGet Packages

Using the NuGet package manager in Visual Studio (Tools → NuGet Package Manager → Manage NuGet Packages for Solution), install the **Microsoft.AspNetCore.OData** NuGet package.

#### 3. Model Class Creation:

Create a model class named **OrdersDetails.cs** in the server-side **Models** folder to represent the order data.

#### ORDERSDETAILS.CS

```
using System.ComponentModel.DataAnnotations;
namespace ODataV4Adaptor.Server.Models
{
```

```

public class OrdersDetails
{
    public static List<OrdersDetails> order = new List<OrdersDetails>();
    public OrdersDetails()
    {
    }
    public OrdersDetails(
        int OrderID, string CustomerId, int EmployeeId, string ShipCountry)
    {
        this.OrderID = OrderID;
        this.CustomerID = CustomerId;
        this.EmployeeID = EmployeeId;
        this.ShipCountry = ShipCountry;
    }
    public static List<OrdersDetails> GetAllRecords()
    {
        if (order.Count() == 0)
        {
            int code = 10000;
            for (int i = 1; i < 10; i++)
            {
                order.Add(new OrdersDetails(code + 1, "ALFKI", i + 0, "Denmark"));
                order.Add(new OrdersDetails(code + 2, "ANATR", i + 2, "Brazil"));
                order.Add(new OrdersDetails(code + 3, "ANTON", i + 1, "Germany"));
                order.Add(new OrdersDetails(code + 4, "BLONP", i + 3, "Austria"));
                order.Add(new OrdersDetails(code + 5, "BOLID", i + 4, "Switzerland"));
                code += 5;
            }
        }
        return order;
    }
    [Key]
    public int? OrderID { get; set; }
    public string? CustomerID { get; set; }
    public int? EmployeeID { get; set; }
    public string? ShipCountry { get; set; }
}

```

#### 4. Build the Entity Data Model

To construct the Entity Data Model for your OData service, utilize the `ODataConventionModelBuilder` to define the model's structure. Start by creating an instance of the `ODataConventionModelBuilder`, then register the entity set **Orders** using the `EntitySet<T>` method, where `OrdersDetails` represents the CLR type containing order details.

```
`cs
```

```
// Create an ODataConventionModelBuilder to build the OData model
```

```
var modelBuilder = new ODataConventionModelBuilder();
```

```
// Register the "Orders" entity set with the OData model builder
```

```
modelBuilder.EntitySet<OrdersDetails>("Orders");
```

```
`
```

## 5. Register the OData Services

Once the Entity Data Model is built, you need to register the OData services in your ASP.NET Core application. Here's how:

```
`cs
// Add controllers with OData support to the service collection
builder.Services.AddControllers().AddOData(
options => options
.Count()
.AddRouteComponents("odata", modelBuilder.GetEdmModel()));
`
```

## 6. Add controllers

Finally, add controllers to expose the OData endpoints. Here's an example:

```
`cs
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.OData.Query;
using ODataV4Adaptor.Server.Models;
namespace ODataV4Adaptor.Server.Controllers
{
[Route("[controller]")]
[ApiController]
public class OrdersController : ControllerBase
{
/// <summary>
/// Retrieves all orders.
/// </summary>
/// <returns>The collection of orders.</returns>
[HttpGet]
[EnableQuery]
public IActionResult Get()
{
var data = OrdersDetails.GetAllRecords().AsQueryable();
return Ok(data);
}
}
```



```

}
}
`

```

#### 4. Run the Application:

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

After running the application, you can verify that the server-side API controller is successfully returning the order data in the URL(<https://localhost:xxxx/odata/Orders>). Here **xxxx** denotes the port number.

##### *Connecting Syncfusion Angular Grid to an OData service*

To integrate the Syncfusion Grid component into your Angular and ASP.NET Core project using Visual Studio, follow the below steps:

#### 1: Install Syncfusion Package

Open your terminal in the project's client folder and install the required Syncfusion packages using npm:

```
`bash
```

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

```
npm install @syncfusion/ej2-data --save
```

```
`
```

#### 2: Import Grid Module

In the `app.module.ts` file, import the **GridModule** from the `@syncfusion/ej2-angular-grids` package:

##### APP.MODULE.TS

```
{% raw %}
import { GridModule } from '@syncfusion/ej2-angular-grids';
@NgModule({
  imports: [
    // Other imports...
    GridModule,
  ],
})
export class AppModule { }
{% endraw %}
```

#### Step 3: Adding CSS reference

Include the necessary CSS files in your `styles.css` file to style the Syncfusion Angular component:

##### 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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
```

```
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

#### Step 4: Adding Syncfusion Component

In your component file (e.g., app.component.ts), import **DataManager** and **ODataV4Adaptor** from **@syncfusion/ej2-data**. Create a **DataManager** instance specifying the URL of your API endpoint(<https://localhost:xxxx/odata/Orders>) using the **url** property and set the adaptor **ODataV4Adaptor**.

#### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/odata/Orders', // Here xxxx represents the port
      number
    });
    adaptor: new ODataV4Adaptor()
  };
}
```

#### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' width='120'
      textAlign='Right' isPrimaryKey="true"></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width='160'></e-
      column>
    <e-column field='EmployeeID' headerText='Employee ID' width='150'></e-
      column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
      column>
  </e-columns>
</ejs-grid>
```

Replace <https://localhost:xxxx/odata/Orders> with the actual **URL** of your API endpoint that provides the data in a consumable format (e.g., JSON).

Run the application in Visual Studio. It will be accessible on a URL like **<https://localhost:xxxx>**.

Ensure your API service is configured to handle CORS (Cross-Origin Resource Sharing) if necessary.

`cs

[program.cs]

```
builder.Services.AddCors(options =>
{
options.AddDefaultPolicy(builder =>
{
builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
});
});
var app = builder.Build();
app.UseCors();
`
```

#### Handling searching operation

To enable search operations in your web application using OData, you first need to configure the OData support in your service collection. This involves adding the **Filter** method within the OData setup, allowing you to filter data based on specified criteria. Once enabled, clients can utilize the **\$filter** query option in their requests to search for specific data entries.

#### PROGRAM.CS

```
// Create a new instance of the web application builder
var builder = WebApplication.CreateBuilder(args);
// Create an ODataConventionModelBuilder to build the OData model
var modelBuilder = new ODataConventionModelBuilder();
// Register the "Orders" entity set with the OData model builder
modelBuilder.EntitySet<OrdersDetails>("Orders");
// Add services to the container.
// Add controllers with OData support to the service collection
builder.Services.AddControllers().AddOData(
options => options
.Count()
.Filter() //searching
.AddRouteComponents("odata", modelBuilder.GetEdmModel()));
```

#### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
selector: 'app-root',
templateUrl: './app.component.html'
})
export class AppComponent {
@ViewChild('grid')
public grid?: GridComponent;
public data?: DataManager;
public toolbar?: ToolbarItems[];
ngOnInit(): void {
```

```

this.data = new DataManager({
url: 'https://localhost:xxxx/odata/Orders',
adaptor: new ODataV4Adaptor()
});
}
this.toolbar = ['Search'];
}

```

#### APP.COMPONENT.HTML

```

<ejs-grid #grid [dataSource]='data' [toolbar]='toolbar'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='EmployeeID' headerText='Employee ID' width='150'></e-
column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>

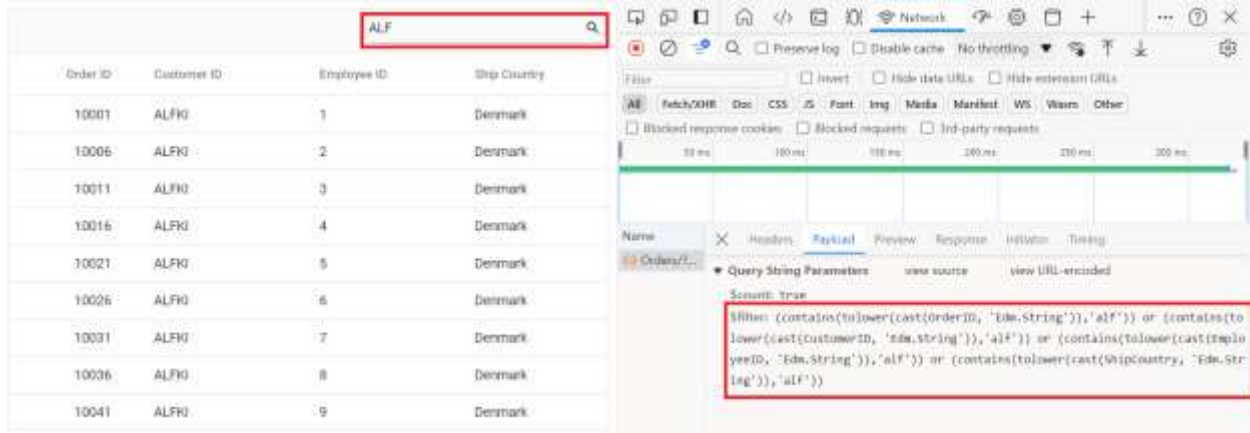
```

#### APP.MODULE.TS

```

import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { GridModule, ToolbarService } from '@syncfusion/ej2-angular-grids';
@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, HttpClientModule,
AppRoutingModule,
GridModule
],
providers: [ToolbarService],
bootstrap: [AppComponent]
})
export class AppModule { }

```



### Handling filtering operation

To enable filter operations in your web application using OData, you first need to configure the OData support in your service collection. This involves adding the **Filter** method within the OData setup, allowing you to filter data based on specified criteria. Once enabled, clients can utilize the **\$filter** query option in your requests to filter for specific data entries.

### PROGRAM.CS

```
// Create a new instance of the web application builder
var builder = WebApplication.CreateBuilder(args);
// Create an ODataConventionModelBuilder to build the OData model
var modelBuilder = new ODataConventionModelBuilder();
// Register the "Orders" entity set with the OData model builder
modelBuilder.EntitySet<OrdersDetails>("Orders");
// Add services to the container.
// Add controllers with OData support to the service collection
builder.Services.AddControllers().AddOData(
options => options
.Count()
.Filter() // filtering
.AddRouteComponents("odata", modelBuilder.GetEdmModel()));
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/odata/Orders', // Replace your hosted link
      adaptor: new ODataV4Adaptor()
    });
  }
}
```

```
}
```

**APP.COMPONENT.HTML**

```
<ejs-grid #grid [dataSource]='data' allowFiltering="true" >
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='EmployeeID' headerText='Employee ID' width='150'></e-
column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>
```

**APP.MODULE.TS**

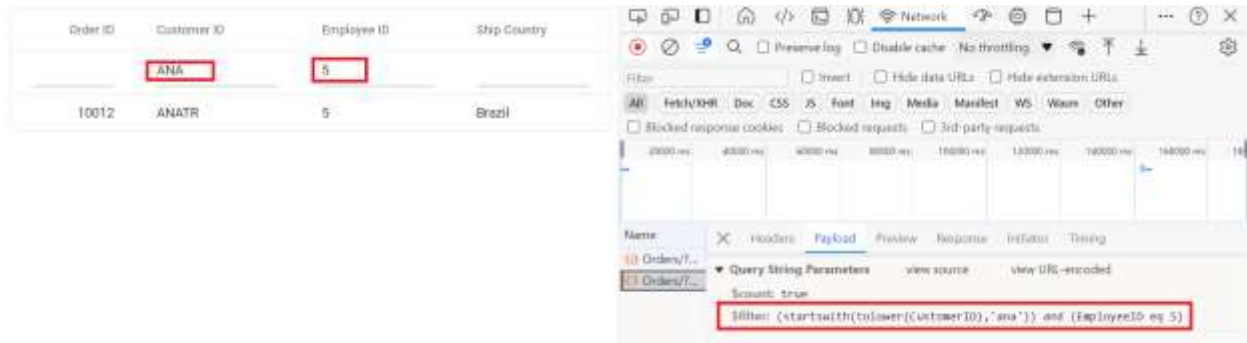
```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { FilterService, GridModule } from '@syncfusion/ej2-angular-grids';
@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, HttpClientModule,
AppRoutingModule,
GridModule
],
providers: [FilterService],
bootstrap: [AppComponent]
})
export class AppModule { }
```

**Single column filtering**

The screenshot displays a web application with a grid of data. The grid has four columns: Order ID, Customer ID, Employee ID, and Ship Country. The Customer ID column is filtered with the value 'ANA'. The grid shows 10 rows of data, all with 'ANATR' in the Customer ID column. To the right, the browser's developer tools are open, showing the Network tab. A request to 'Orders/...' is selected, and the Query String Parameters are visible. The filter is applied as a query string parameter: 'Search: true' and '\$filter: (startswith(tolower(customerID), 'ana'))'.

Order ID	Customer ID	Employee ID	Ship Country
	ANA		
10002	ANATR	3	Brazil
10007	ANATR	4	Brazil
10012	ANATR	5	Brazil
10017	ANATR	6	Brazil
10022	ANATR	7	Brazil
10027	ANATR	8	Brazil
10032	ANATR	9	Brazil
10037	ANATR	10	Brazil
10042	ANATR	11	Brazil

## Multi column filtering

*Handling sorting operation*

To enable sorting operations in your web application using OData, you first need to configure the OData support in your service collection. This involves adding the **OrderBy** method within the OData setup, allowing you to sort data based on specified criteria. Once enabled, clients can utilize the **\$orderby** query option in their requests to sort data entries according to desired attributes.

**PROGRAM.CS**

```
// Create a new instance of the web application builder
var builder = WebApplication.CreateBuilder(args);
// Create an ODataConventionModelBuilder to build the OData model
var modelBuilder = new ODataConventionModelBuilder();
// Register the "Orders" entity set with the OData model builder
modelBuilder.EntitySet<OrdersDetails>("Orders");
// Add services to the container.
// Add controllers with OData support to the service collection
builder.Services.AddControllers().AddOData(
options => options
.Count()
.OrderBy() // sorting
.AddRouteComponents("odata", modelBuilder.GetEdmModel()));
```

**APP.COMPONENT.TS**

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
public data?: DataManager;
@ViewChild('grid')
public grid?: GridComponent;
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/odata/Orders', // Replace your hosted link
adaptor: new ODataV4Adaptor()
});
}
}
```

**APP.COMPONENT.HTML**

```
<ejs-grid #grid [dataSource]='data' allowSorting="true" >
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='EmployeeID' headerText='Employee ID' width='150'></e-
column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>
```

**APP.MODULE.TS**

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SortService, GridModule } from '@syncfusion/ej2-angular-grids';
@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, HttpClientModule,
AppRoutingModule,
GridModule
],
providers: [SortService],
bootstrap: [AppComponent]
})
export class AppModule { }
```

*Single column sorting*

Order ID	Customer ID	↑ Employee ID	Ship Country
10001	ALFKI	1	Denmark
10006	ALFKI	2	Denmark
10011	ALFKI	3	Denmark
10016	ALFKI	4	Denmark
10021	ALFKI	5	Denmark
10026	ALFKI	6	Denmark
10031	ALFKI	7	Denmark
10036	ALFKI	8	Denmark

*Multi column sorting*



Order ID	Customer ID	Employee ID	Ship Country
10041	ALFKI	9	Denmark
10036	ALFKI	8	Denmark
10031	ALFKI	7	Denmark
10026	ALFKI	6	Denmark
10021	ALFKI	5	Denmark
10016	ALFKI	4	Denmark
10011	ALFKI	3	Denmark
10006	ALFKI	2	Denmark
10001	ALFKI	1	Denmark
10042	ANATR	11	Brazil

### Handling paging operation

To implement paging operations in your web application using OData, you can utilize the **SetMaxTop** method within your OData setup to limit the maximum number of records that can be returned per request. While you configure the maximum limit, clients can utilize the **\$skip** and **\$top** query options in their requests to specify the number of records to skip and the number of records to take, respectively.

### PROGRAM.CS

```
// Create a new instance of the web application builder
var builder = WebApplication.CreateBuilder(args);
// Create an ODataConventionModelBuilder to build the OData model
var modelBuilder = new ODataConventionModelBuilder();
// Register the "Orders" entity set with the OData model builder
modelBuilder.EntitySet<OrdersDetails>("Orders");
// Add services to the container.
// Add controllers with OData support to the service collection
var recordCount= OrdersDetails.GetAllRecords().Count;
builder.Services.AddControllers().AddOData(
options => options
.Count()
.SetMaxTop(recordCount)
.AddRouteComponents(
"odata",
modelBuilder.GetEdmModel()));
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
public data?: DataManager;
@ViewChild('grid')
public grid?: GridComponent;
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/odata/Orders', // Replace your hosted link
adaptor: new ODataV4Adaptor()
});
}
```

```
});  
}  
}
```

### **APP.COMPONENT.HTML**

```
<ejs-grid #grid [dataSource]='data' allowPaging="true" >  
<e-columns>  
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true  
width='150'></e-column>  
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-  
column>  
<e-column field='EmployeeID' headerText='Employee ID' width='150'></e-  
column>  
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-  
column>  
</e-columns>  
</ejs-grid>
```

### **APP.MODULE.TS**

```
import { HttpClientModule } from '@angular/common/http';  
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { AppRoutingModuleModule } from './app-routing.module';  
import { AppComponent } from './app.component';  
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids';  
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule, HttpClientModule,  
    AppRoutingModuleModule,  
    GridModule  
  ],  
  providers: [PageService],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

Order ID	Customer Name	Employee ID	Ship Country
10013	ANTON	4	Germany
10014	BONAP	5	Austria
10015	BOLID	7	Switzerland
10016	ALFKI	4	Denmark
10017	ANATR	6	Brazil
10018	ANTON	5	Germany
10019	BONAP	7	Austria
10020	BOLID	8	Switzerland
10021	ALFKI	5	Denmark
10022	ANATR	7	Brazil
10023	ANTON	6	Germany
10024	BONAP	8	Austria

### Handling CRUD operations

To manage CRUD (Create, Read, Update, Delete) operations using the ODataV4Adaptor, follow the provided guide for configuring the Syncfusion Grid for [editing](#) and utilize the sample implementation of the `OrdersController` in your server application. This controller handles HTTP requests for CRUD operations such as GET, POST, PATCH, and DELETE.

To enable CRUD operations in the Syncfusion Grid component within an Angular application, follow the below steps:

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { GridComponent, ToolbarItems, EditSettingsModel } from '@syncfusion/ej2-angular-grids';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('grid')
  public grid?: GridComponent;
  public data?: DataManager;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/odata/Orders', // xxxx denotes port number
      adaptor: new ODataV4Adaptor()
    });
    this.editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Normal' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'];
  }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [editSettings]="editSettings"
[toolbar]="toolbar">
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='EmployeeID' headerText='Employee ID' width='150'></e-
column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>
```

### APP.MODULE.TS

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { EditService, GridModule, ToolbarService } from '@syncfusion/ej2-
angular-grids';
@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, HttpClientModule,
AppRoutingModule,
GridModule
],
providers: [EditService, ToolbarService],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Normal/Inline editing is the default edit [mode](#) for the Grid component. To enable CRUD operations, ensure that the [isPrimaryKey](#) property is set to **true** for a specific Grid column, ensuring that its value is unique.

### Insert Record

To insert a new record into your Syncfusion Grid, you can utilize the **HttpPost** method in your server application. Below is a sample implementation of inserting a record using the **OrdersController**:

```
`cs
/// <summary>
/// Inserts a new order to the collection.
/// </summary>
/// <param name="addRecord">The order to be inserted.</param>
/// <returns>It returns the newly inserted record detail.</returns>
```

```
[HttpPost]
```

```
[EnableQuery]
```

```
public IActionResult Post([FromBody] OrdersDetails addRecord)
```

```
{
```

```
if (addRecord == null)
```

```
{
```

```
return BadRequest("Null order");
```

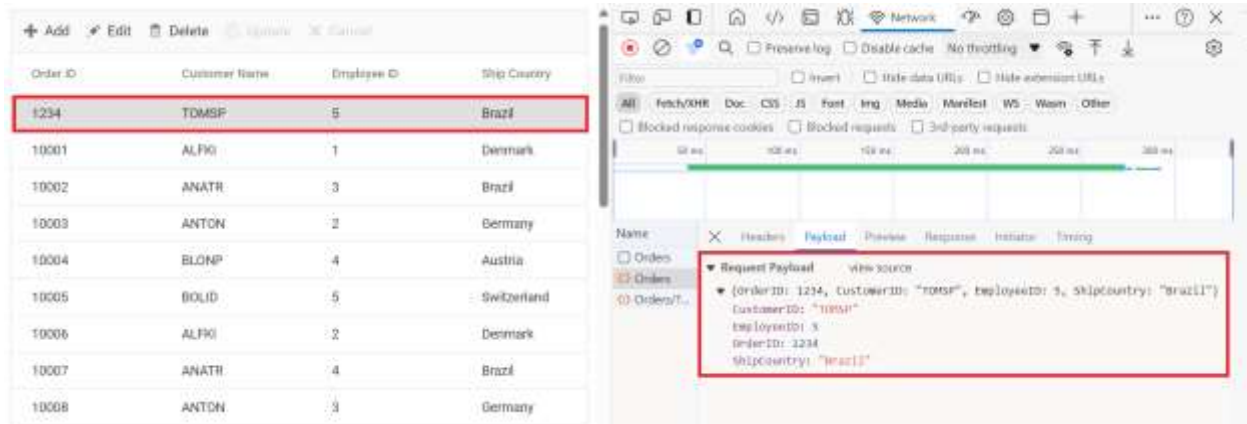
```
}
```

```
OrdersDetails.GetAllRecords().Insert(0, addRecord);
```

```
return Json(addRecord);
```

```
}
```

```
,
```



## Update Record

Updating a record in the Syncfusion Grid can be achieved by utilizing the `HttpPatch` method in your controller. Here's a sample implementation of updating a record:

```
`cs
```

```
/// <summary>
```

```
/// Updates an existing order.
```

```
/// </summary>
```

```
/// <param name="key">The ID of the order to update.</param>
```

```
/// <param name="updateRecord">The updated order details.</param>
```

```
/// <returns>It returns the updated order details.</returns>
```

```
[HttpPatch("{key}")]
```

```
public IActionResult Patch(int key, [FromBody] OrdersDetails updateRecord)
```

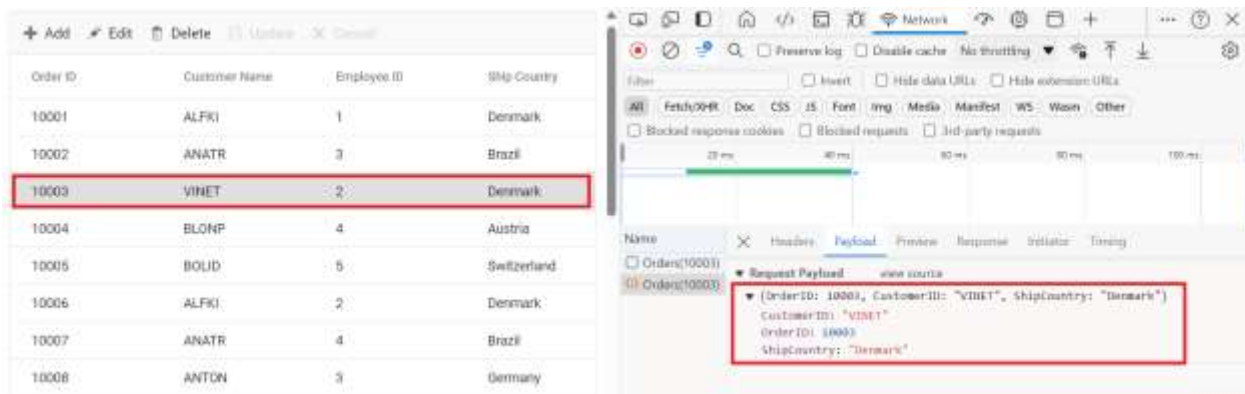
```
{
```

```

if (updateRecord == null)
{
    return BadRequest("No records");
}

var existingOrder = OrdersDetails.GetAllRecords().FirstOrDefault(order => order.OrderID == key);
if (existingOrder != null)
{
    // If the order exists, update its properties
    existingOrder.CustomerID = updateRecord.CustomerID ?? existingOrder.CustomerID;
    existingOrder.EmployeeID = updateRecord.EmployeeID ?? existingOrder.EmployeeID;
    existingOrder.ShipCountry = updateRecord.ShipCountry ?? existingOrder.ShipCountry;
}
return Json(updateRecord);
}

```



### Delete Record

To delete a record from your Syncfusion Grid, you can utilize the `HttpDelete` method in your controller. Below is a sample implementation:

```

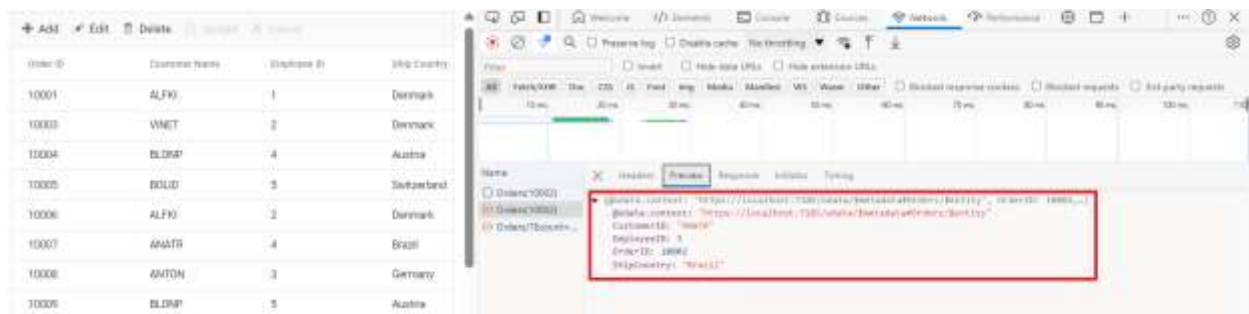
`cs
/// <summary>
/// Deletes an order.
/// </summary>
/// <param name="key">The ID of the order to delete.</param>
/// <returns>It returns the deleted record detail</returns>
[HttpDelete("{key}")]
public IActionResult Delete(int key)

```

```

{
var deleteRecord = OrdersDetails.GetAllRecords().FirstOrDefault(order => order.OrderID == key);
if (deleteRecord != null)
{
OrdersDetails.GetAllRecords().Remove(deleteRecord);
}
return Json(deleteRecord);
}

```



You can find the complete sample for the ODataV4Adaptor in [GitHub](#) link.

### WebApiAdaptor in Syncfusion Angular Grid Component

The **WebApiAdaptor** is an extension of the **ODataAdaptor**, designed to interact with Web APIs created with OData endpoints. This adaptor ensures seamless communication between Syncfusion Grid and OData-endpoint based Web APIs, enabling efficient data retrieval and manipulation. For successful integration, the endpoint must be capable of understanding OData-formatted queries sent along with the request.

To enable the OData query option for a Web API, please refer to the corresponding [documentation](#), which provides detailed instructions on configuring the endpoint to understand OData-formatted queries.

This section describes a step-by-step process for retrieving data service using **WebApiAdaptor**, then binding it to the Angular Grid component to facilitate data and CRUD operations.

#### Creating a Web API service

To configure a server for use with Syncfusion Angular Grid, you need to follow the below steps:

##### 1. Project Creation:

Open Visual Studio and create an Angular and ASP.NET Core project named **WebApiAdaptor**. To create an Angular and ASP.NET Core application, follow the documentation [link](#) for detailed steps.

##### 2. Model Class Creation:

Create a model class named **OrdersDetails.cs** in the server-side **Models** folder to represent the order data.

#### ORDERSDETAILS.CS

```

namespace WebApiAdaptor.Server.Models
{
    public class OrdersDetails
    {
        public static List<OrdersDetails> order = new List<OrdersDetails>();
        public OrdersDetails()
        {
        }
        public OrdersDetails(
            int OrderID, string CustomerId, int EmployeeId, double Freight, bool
            Verified,
            DateTime OrderDate, string ShipCity, string ShipName, string ShipCountry,
            DateTime ShippedDate, string ShipAddress)
        {
            this.OrderID = OrderID;
            this.CustomerID = CustomerId;
            this.EmployeeID = EmployeeId;
            this.Freight = Freight;
            this.ShipCity = ShipCity;
            this.Verified = Verified;
            this.OrderDate = OrderDate;
            this.ShipName = ShipName;
            this.ShipCountry = ShipCountry;
            this.ShippedDate = ShippedDate;
            this.ShipAddress = ShipAddress;
        }
        public static List<OrdersDetails> GetAllRecords()
        {
            if (order.Count() == 0)
            {
                int code = 10000;
                for (int i = 1; i < 10; i++)
                {
                    order.Add(new OrdersDetails(code + 1, "ALFKI", i + 0, 2.3 * i, false, new
                    DateTime(1991, 05, 15), "Berlin", "Simons bistro", "Denmark", new
                    DateTime(1996, 7, 16), "Kirchgasse 6"));
                    order.Add(new OrdersDetails(code + 2, "ANATR", i + 2, 3.3 * i, true, new
                    DateTime(1990, 04, 04), "Madrid", "Queen Cozinha", "Brazil", new
                    DateTime(1996, 9, 11), "Avda. Azteca 123"));
                    order.Add(new OrdersDetails(code + 3, "ANTON", i + 1, 4.3 * i, true, new
                    DateTime(1957, 11, 30), "Cholchester", "Frankenversand", "Germany", new
                    DateTime(1996, 10, 7), "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
                    order.Add(new OrdersDetails(code + 4, "BLONP", i + 3, 5.3 * i, false, new
                    DateTime(1930, 10, 22), "Marseille", "Ernst Handel", "Austria", new
                    DateTime(1996, 12, 30), "Magazinweg 7"));
                    order.Add(new OrdersDetails(code + 5, "BOLID", i + 4, 6.3 * i, true, new
                    DateTime(1953, 02, 18), "Tsawassen", "Hanari Carnes", "Switzerland", new
                    DateTime(1997, 12, 3), "1029 - 12th Ave. S.));
                    code += 5;
                }
            }
            return order;
        }
        public int? OrderID { get; set; }
        public string? CustomerID { get; set; }
        public int? EmployeeID { get; set; }
        public double? Freight { get; set; }
    }
}

```



```
public string? ShipCity { get; set; }
public bool? Verified { get; set; }
public DateTime OrderDate { get; set; }
public string? ShipName { get; set; }
public string? ShipCountry { get; set; }
public DateTime ShippedDate { get; set; }
public string? ShipAddress { get; set; }
}
}
```

### 3. API Controller Creation:

Create a file named `OrdersController.cs` under the **Controllers** folder. This controller will handle data communication with the Angular Grid component. Implement the **Get** method in the controller to return the data in JSON format, including the **Items** and **Count** properties as required by `WebApiAdaptor`.

#### ORDERSCONTROLLER.CS

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebApiAdaptor.Server.Models;
namespace WebApiAdaptor.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class OrdersController : ControllerBase
    {
        [HttpGet]
        public object Get()
        {
            var data = OrdersDetails.GetAllRecords().ToList();
            return new { Items = data, Count = data.Count() };
        }
    }
}
```

### 4. Run the Application:

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

After running the application, you can verify that the server-side API controller is successfully returning the order data in the URL(<https://localhost:xxxx/api/Orders>). Here **xxxx** denotes the port number.

The screenshot shows a web browser with the address bar displaying `https://localhost:xxxx/api/Orders`. The page content is a JSON array of four order objects, each with fields like `orderID`, `customerID`, `employeeID`, `freight`, `shipCity`, `verified`, `orderDate`, `shipName`, `shipCountry`, `shippedDate`, and `shipAddress`.

```

1 {
2   "items": [
3     {
4       "orderID": 10001,
5       "customerID": "ALFKI",
6       "employeeID": 1,
7       "freight": 2.3,
8       "shipCity": "Berlin",
9       "verified": false,
10      "orderDate": "1991-05-15T00:00:00",
11      "shipName": "Simons bistro",
12      "shipCountry": "Denmark",
13      "shippedDate": "1996-07-16T00:00:00",
14      "shipAddress": "Kirchgasse 6"
15    },
16    {
17      "orderID": 10002,
18      "customerID": "ANATR",
19      "employeeID": 3,
20      "freight": 3.3,
21      "shipCity": "Madrid",
22      "verified": true,
23      "orderDate": "1990-04-04T00:00:00",
24      "shipName": "Queen Cozinha",
25      "shipCountry": "Brazil",
26      "shippedDate": "1996-09-11T00:00:00",
27      "shipAddress": "Avda. Azteca 123"
28    },
29    {
30      "orderID": 10003,
31      "customerID": "ANTON",
32      "employeeID": 2,
33      "freight": 4.3,
34      "shipCity": "Cholchester",
35      "verified": true,
36      "orderDate": "1957-11-30T00:00:00",
37      "shipName": "Frankenversand",
38      "shipCountry": "Germany",
39      "shippedDate": "1996-10-07T00:00:00",
40      "shipAddress": "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"
41    },
42    {
43      "orderID": 10004,
44      "customerID": "BLONP",
45      "employeeID": 4,

```

### Connecting Syncfusion Angular Grid to an API service

To integrate the Syncfusion Grid component into your Angular and ASP.NET Core project using Visual Studio, follow the below steps:

#### 1: Install Syncfusion Package

Open your terminal in the project's root directory of client folder and install the required Syncfusion packages using npm:

```
`bash
```

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

```
npm install @syncfusion/ej2-data --save
```

```
`
```

#### 2: Import Grid Module

In the `app.module.ts` file, import the **GridModule** from the `@syncfusion/ej2-angular-grids` package:

#### Step 3: Adding CSS reference

Include the necessary CSS files in your `styles.css` file to style the Syncfusion Angular components:

#### 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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';

```

```
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

#### Step 4: Adding Syncfusion Component

In your component file (e.g., app.component.ts), import **DataManager** and **WebApiAdaptor** from **@syncfusion/ej2-data**. Create a **DataManager** instance specifying the URL of your API endpoint(<https://localhost:xxxx/api/Orders>) using the **url** property and set the adaptor **WebApiAdaptor**.

#### APP.COMPONENT.TS

```
{% raw %}
import { Component, ViewChild } from '@angular/core';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Orders', // Here xxxx represents the port
      number
    });
    adaptor: new WebApiAdaptor()
  });
}
}{% endraw %}
```

#### APP.COMPONENT.HTML

```
{% raw %}
<ejs-grid #grid [dataSource]='data'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' width='120'
      textAlign='Right' isPrimaryKey="true"></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width='160'></e-
      column>
    <e-column field='ShipCity' headerText='Ship City' width='150'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
      column>
  </e-columns>
</ejs-grid>
}{% endraw %}
```

Replace <https://localhost:xxxx/api/Orders> with the actual URL of your API endpoint that provides the data in a consumable format (e.g., JSON).

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

Ensure your API service is configured to handle CORS (Cross-Origin Resource Sharing) if necessary.

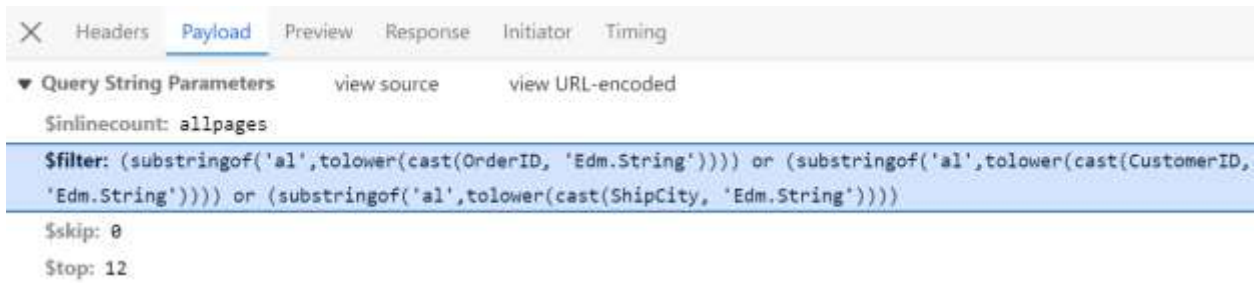
```
`cs
[program.cs]
builder.Services.AddCors(options =>
{
options.AddDefaultPolicy(builder =>
{
builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
});
});
var app = builder.Build();
app.UseCors();
`
```

You can find the complete sample for the WebApiAdaptor in [GitHub](#) link.

Order ID	Customer ID	Ship City	Ship Country	
10001	ALFKI	Berlin	Denmark	
10002	ANATR	Madrid	Brazil	
10003	ANTON	Cholchester	Germany	
10004	BLONP	Marseille	Austria	
10005	BOLID	Tsawassen	Switzerland	
10006	ALFKI	Berlin	Denmark	
10007	ANATR	Madrid	Brazil	
10008	ANTON	Cholchester	Germany	
10009	BLONP	Marseille	Austria	
10010	BOLID	Tsawassen	Switzerland	
10011	ALFKI	Berlin	Denmark	
10012	ANATR	Madrid	Brazil	
10013	ANTON	Cholchester	Germany	

### Handling searching operation

To handle search operation, implement search logic on the server side according to the received OData-formatted query.



### ORDERSCONTROLLER.CS

```
// GET: api/Orders
[HttpGet]
public object Get()
{
    var queryString = Request.Query;
    var data = OrdersDetails.GetAllRecords().ToList();
    string filter = queryString["$filter"];
    if (filter != null)
    {
        var filters = filter.Split(new string[] { " and " },
            StringSplitOptions.RemoveEmptyEntries);
        foreach (var filterItem in filters)
        {
            if (filterItem.Contains("substringof"))
            {
                // Perform Searching
                var searchParts = filterItem.Split('(', ')', '\\');
                var searchValue = searchParts[3];
                // Apply the search value to all searchable fields
                data = data.Where(cust =>
                    cust.OrderID.ToString().Contains(searchValue) ||
                    cust.CustomerID.ToLower().Contains(searchValue) ||
                    cust.ShipCity.ToLower().Contains(searchValue)
                ).ToList();
                // Add conditions for other searchable fields as needed
            }
        }
    }
    else
    {
        // Perform filtering
    }
}
return new { Items = data, Count = data.Count() };
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
```

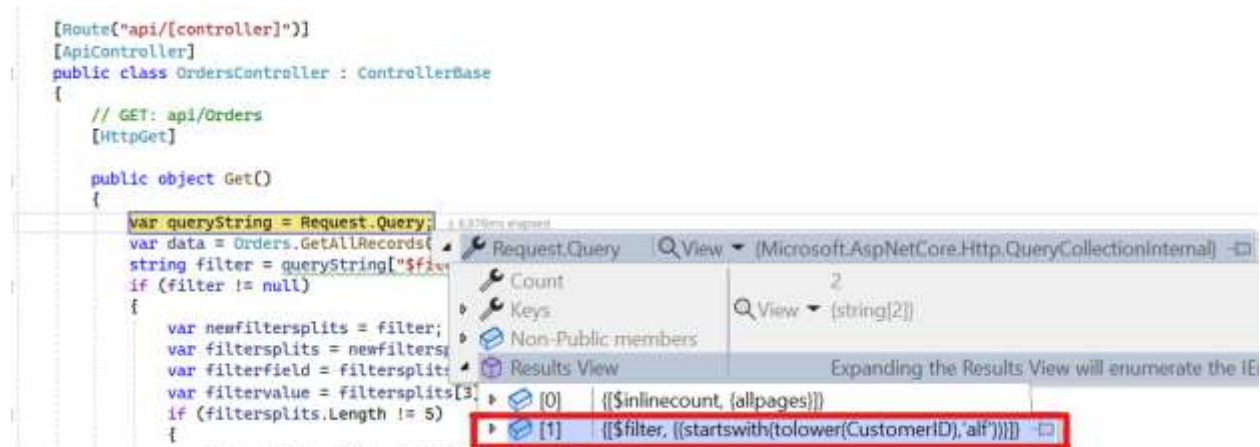
```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('grid')
  public grid?: GridComponent;
  public data?: DataManager;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'api/Orders',
      adaptor: new WebApiAdaptor()
    });
  }
  this.toolbar = ['Search'];
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [toolbar]='toolbar' height='320'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-column>
  </e-columns>
</ejs-grid>
```

### Handling filtering operation

To handle filter operations, ensure that your Web API endpoint supports filtering based on OData-formatted queries. Implement the filtering logic on the server-side as shown in the provided code snippet.



### ORDERSCONTROLLER.CS

```
// GET: api/Orders
[HttpGet]
public object Get()
{
    var queryString = Request.Query;
    var data = Orders.GetAllRecords().ToList();
    string filter = queryString["$filter"];
    if (filter != null)
    {
        var filters = filter.Split(new string[] { " and " },
            StringSplitOptions.RemoveEmptyEntries);
        foreach (var filterItem in filters)
        {
            var filterfield = "";
            var filtervalue = "";
            var filterParts = filterItem.Split('(', ')', '\\');
            if (filterParts.Length != 9)
            {
                var filterValueParts = filterParts[1].Split();
                filterfield = filterValueParts[0];
                filtervalue = filterValueParts[2];
            }
            else
            {
                filterfield = filterParts[3];
                filtervalue = filterParts[5];
            }
            switch (filterfield)
            {
                case "OrderID":
                    data = (from cust in data
                        where cust.OrderID.ToString() == filtervalue.ToString()
                        select cust).ToList();
                    break;
                case "CustomerID":
                    data = (from cust in data
                        where cust.CustomerID.ToLower().StartsWith(filtervalue.ToString())
                        select cust).ToList();
                    break;
                case "ShipCity":
                    data = (from cust in data
                        where cust.ShipCity.ToLower().StartsWith(filtervalue.ToString())
                        select cust).ToList();
                    break;
            }
        }
        return new { Items = data, Count = data.Count() };
    }
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
```

```

selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Orders', // Replace your hosted link
      adaptor: new WebApiAdaptor()
    });
  }
}

```

### APP.COMPONENT.HTML

```

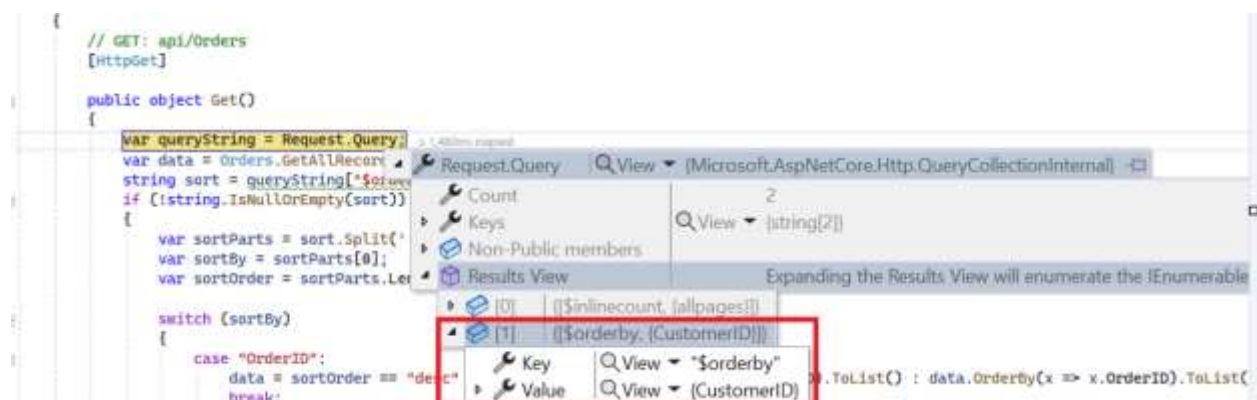
<ejs-grid #grid [dataSource]='data' allowFiltering="true" height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
      column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
      column>
  </e-columns>
</ejs-grid>

```

### Handling sorting operation

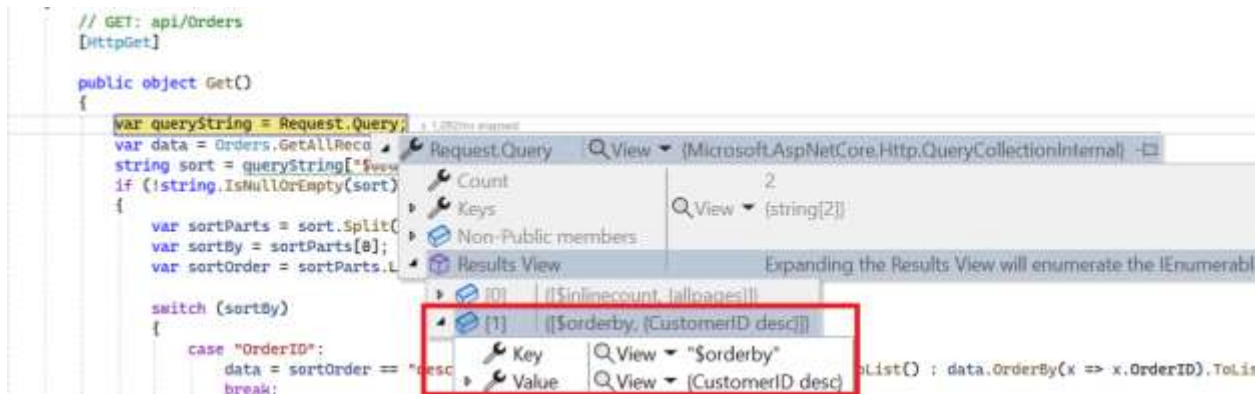
To handle sorting action, implement sorting logic on the server-side according to the received OData-formatted query.

#### Ascending Sorting



#### Descending Sorting





### ORDERSCONTROLLER.CS

```
// GET: api/Orders
[HttpGet]
public object Get()
{
    var queryString = Request.Query;
    var data = OrdersDetails.GetAllRecords().ToList();
    string sort = queryString["$orderby"]; //sorting
    if (!string.IsNullOrEmpty(sort))
    {
        var sortConditions = sort.Split(',');
        var orderedData = data.OrderBy(x => 0); // Start with a stable sort
        foreach (var sortCondition in sortConditions)
        {
            var sortParts = sortCondition.Trim().Split(' ');
            var sortBy = sortParts[0];
            var sortOrder = sortParts.Length > 1 && sortParts[1].ToLower() == "desc";
            switch (sortBy)
            {
                case "OrderID":
                    orderedData = sortOrder ? orderedData.OrderByDescending(x => x.OrderID) :
                    orderedData.OrderBy(x => x.OrderID);
                    break;
                case "CustomerID":
                    orderedData = sortOrder ? orderedData.OrderByDescending(x => x.CustomerID) :
                    orderedData.OrderBy(x => x.CustomerID);
                    break;
                case "ShipCity":
                    orderedData = sortOrder ? orderedData.OrderByDescending(x => x.ShipCity) :
                    orderedData.OrderBy(x => x.ShipCity);
                    break;
            }
        }
        data = orderedData.ToList();
    }
    return new { Items = data, Count = data.Count() };
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
```

```
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Orders', // Replace your hosted link
      adaptor: new WebApiAdaptor()
    });
  }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' allowSorting="true" height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
      column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
      column>
  </e-columns>
</ejs-grid>
```

### Handling paging operation

Implement paging logic on the server-side according to the received OData-formatted query. Ensure that the endpoint supports paging based on the specified criteria.

The screenshot shows a C# code file with two GET endpoints. The first endpoint is a simple GET, and the second is a GET with an ID parameter. The Results View for the second endpoint is expanded, showing a table with three rows of data. The first row is for the inline count, the second for the skip value, and the third for the stop value.

Index	Value
[0]	{{Inlinecount, allpages}}
[1]	{{Skip, 0}}
[2]	{{Stop, 12}}

### ORDERSCONTROLLER.CS

```
// GET: api/Orders
[HttpGet]
public object Get()
{
  var queryString = Request.Query;
```

```

var data = Orders.GetAllRecords().ToList();
// Perform Paging operation
int skip = Convert.ToInt32(queryString["$skip"]);
int take = Convert.ToInt32(queryString["$top"]);
return take != 0 ? new { Items = data.Skip(skip).Take(take).ToList(), Count = data.Count() } : new { Items = data, Count = data.Count() };
}

```

### APP.COMPONENT.TS

```

import { Component, ViewChild } from '@angular/core';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Orders', // Replace your hosted link
      adaptor: new WebApiAdaptor()
    });
  }
}

```

### APP.COMPONENT.HTML

```

<ejs-grid #grid [dataSource]='data' allowPaging="true" height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-column>
  </e-columns>
</ejs-grid>

```

#### Handling CRUD operations

To manage CRUD (Create, Read, Update, Delete) operations using the WebApiAdaptor, follow the provided guide for configuring the Syncfusion Grid for [editing](#) and utilize the sample implementation of the **OrdersController** in your server application. This controller handles HTTP requests for CRUD operations such as GET, POST, PUT, and DELETE.

To enable CRUD operations in the Syncfusion Grid component within an Angular application, follow the below steps:

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { GridComponent, ToolbarItems, EditSettingsModel } from
 '@syncfusion/ej2-angular-grids';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html'
})
export class AppComponent {
  @ViewChild('grid')
  public grid?: GridComponent;
  public data?: DataManager;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Orders',
      adaptor: new WebApiAdaptor()
    });
    this.editSettings = { allowEditing: true, allowAdding: true, allowDeleting:
      true, mode: 'Normal' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'];
  }
}
```

#### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [editSettings]="editSettings"
 [toolbar]="toolbar" height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
    width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
    column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
    textAlign='Right'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
    column>
  </e-columns>
</ejs-grid>
```

Normal/Inline editing is the default edit [mode](#) for the Grid component. To enable CRUD operations, ensure that the [isPrimaryKey](#) property is set to **true** for a specific Grid column, ensuring that its value is unique.

#### Insert Record

To insert a new record into your Syncfusion Grid, you can utilize the `HttpPost` method in your server application. Below is a sample implementation of inserting a record using the **OrdersController**:



```
`cs
```

```
// POST: api/Orders
```

```
[HttpPost]
```

```
/// <summary>
```

```
/// Inserts a new data item into the data collection.
```

```
/// </summary>
```

```
/// <param name="newRecord">It holds new record detail which is need to be inserted.</param>
```

```
/// <returns>Returns void</returns>
```

```
public void Post([FromBody] OrdersDetails newRecord)
```

```
{
```

```
// Insert a new record into the OrdersDetails model
```

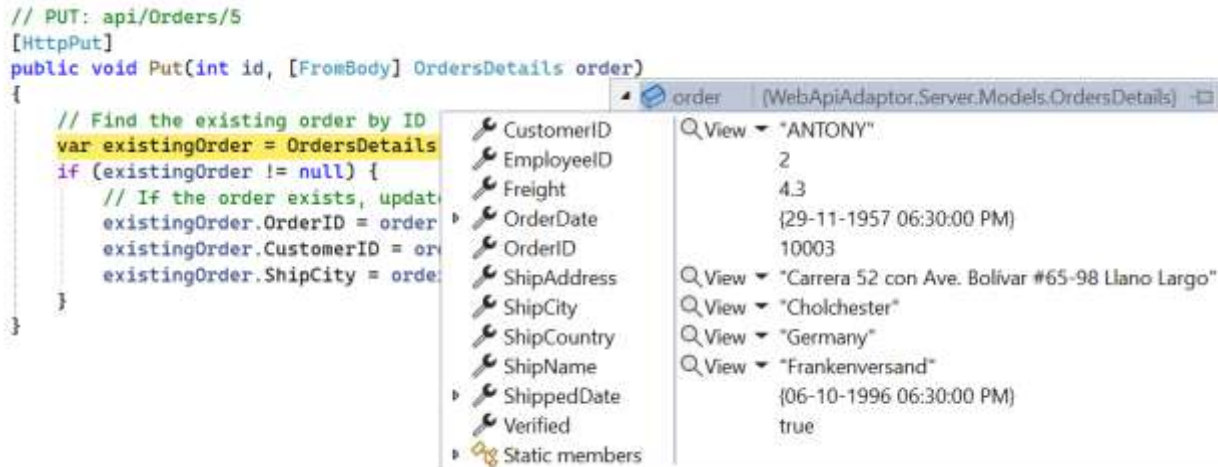
```
OrdersDetails.GetAllRecords().Insert(0, newRecord);
```

```
}
```

```
,
```

## Update Record

Updating a record in the Syncfusion Grid can be achieved by utilizing the `HttpPut` method in your controller. Here's a sample implementation of updating a record:



```
`cs
```

```
// PUT: api/Orders/5
```

```
[HttpPut]
```

```
/// <summary>
```

```
/// Update a existing data item from the data collection.
```

```
/// </summary>
```

```
/// <param name="updatedOrder">It holds updated record detail which is need to be updated.</param>
```

```
/// <returns>Returns void</returns>
```

```
public void Put(int id, [FromBody] OrdersDetails updatedOrder)
```

```
{
```

```
// Find the existing order by ID
```

```
var existingOrder = OrdersDetails.GetAllRecords().FirstOrDefault(o => o.OrderID == id);
```

```
if (existingOrder != null)
```

```
{
```

```
// If the order exists, update its properties
```

```
existingOrder.OrderID = updatedOrder.OrderID;
```

```
existingOrder.CustomerID = updatedOrder.CustomerID;
```

```
existingOrder.ShipCity = updatedOrder.ShipCity;
```

```
existingOrder.ShipCountry = updatedOrder.ShipCountry;
```

```
}
```

```
}
```

```
,
```

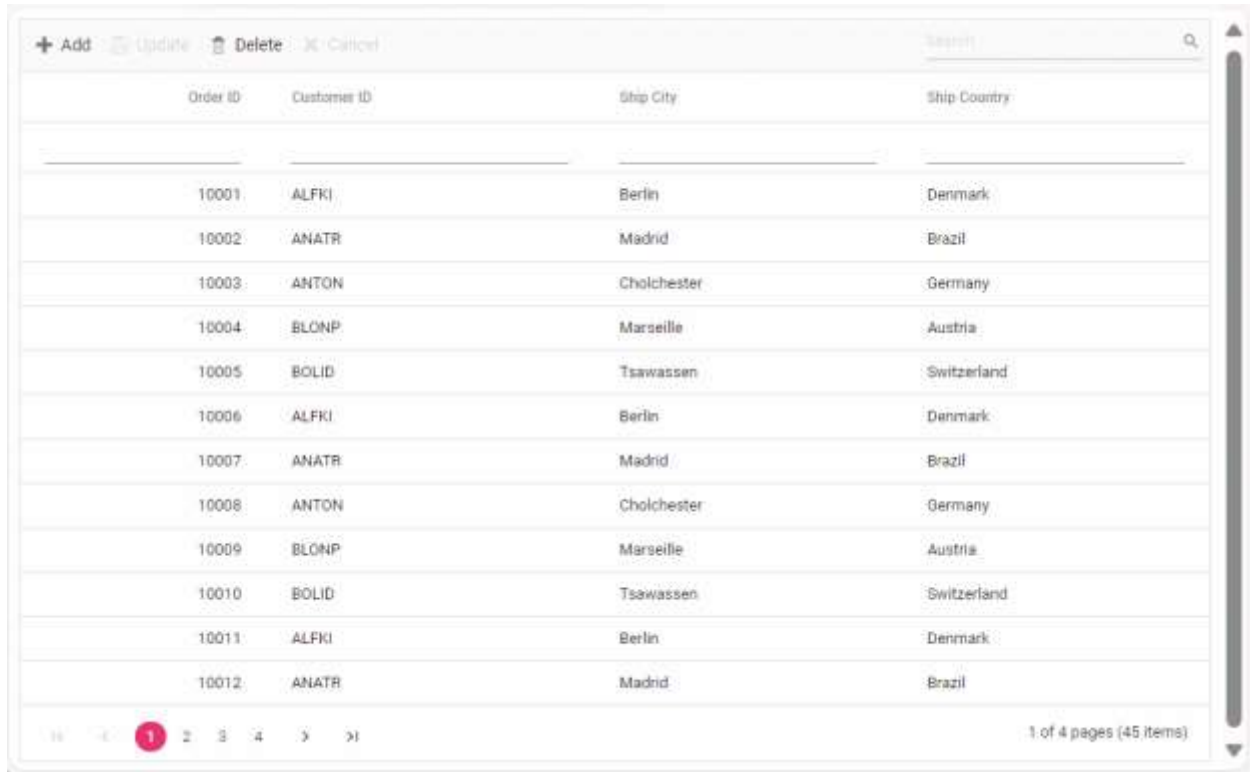
## Delete Record

To delete a record from your Syncfusion Grid, you can utilize the `HttpDelete` method in your controller. Below is a sample implementation:

```
// DELETE: api/5
[HttpDelete("{id}")]
public void Delete(int id)
{
    // Find the order to remove by ID
    var orderToRemove = OrdersDetails.GetAllRecords().FirstOrDefault(order => order.OrderID == id);
    // If the order exists, remove it
    if (orderToRemove != null)
    {
        OrdersDetails.GetAllRecords().Remove(orderToRemove);
    }
}
```

`cs

```
// DELETE: api/5
[HttpDelete("{id}")]
/// <summary>
/// Remove a specific data item from the data collection.
/// </summary>
/// <param name="key">It holds specific record detail id which is need to be removed.</param>
/// <returns>Returns void</returns>
public void Delete(int key)
{
    // Find the order to remove by ID
    var orderToRemove = OrdersDetails.GetAllRecords().FirstOrDefault(order => order.OrderID == key);
    // If the order exists, remove it
    if (orderToRemove != null)
    {
        OrdersDetails.GetAllRecords().Remove(orderToRemove);
    }
}
`
```



Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil

You can find the complete sample for the WebApiAdaptor in [GitHub](#) link.

### Connecting GraphQL Service with Angular Grid Component

GraphQL is a powerful query language for APIs, designed to provide a more efficient alternative to traditional REST APIs. It allows you to precisely fetch the data you need, reducing over-fetching and under-fetching of data. GraphQL provides a flexible and expressive syntax for querying, enabling clients to request only the specific data they require.

Syncfusion's Grid component seamlessly integrates with GraphQL servers using the [GraphQLAdaptor](#) in the [DataManager](#). This specialized adaptor simplifies the interaction between the Syncfusion Grid and GraphQL servers, allowing efficient data retrieval with support for various operations like CRUD (Create, Read, Update, Delete), paging, sorting, and filtering.

This section describes a step-by-step process for retrieving data from GraphQL service using [GraphQLAdaptor](#), then binding it to the Angular Grid component to facilitate data and CRUD operations.

#### Configure GraphQL Server

To configure a GraphQL server with Syncfusion Angular Grid, you need to follow the below steps:

##### Step 1: Create Service for GraphQL

- Create a new folder named **GraphQLServer** specifically for your GraphQL server.
- Install the [graph pack](#) npm package. Open your terminal and navigate to the server folder, then run:

```
`bash
```

```
npm i graphpack
```



,

- To utilize Syncfusion's **ej2-data** package, you need to include it as a dependency in your project's **package.json** file. Here's how you can mention it in the configuration:

```
`json
{
  "name": "graphql-server",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "dev": "graphpack --port 4205",
    "build": "graphpack build"
  },
  "devDependencies": {
    "graphpack": "^1.0.9"
  },
  "dependencies": {
    "@syncfusion/ej2-data": "24.1.41"
  }
}
```

,

- Create a database file (src/db.js) to store your data.

```
`js
export let OrderData = [
  {
    OrderID: 10248, CustomerID: 'VINET', EmployeeID: 5, OrderDate: new Date("07 12 1996 02:00:23"),
    ShipName: 'Vins et alcools Chevalier', ShipCity: 'Reims', ShipAddress: '59 rue de l Abbaye',
    ShipRegion: 'CJ', ShipPostalCode: '51100', ShipCountry: 'France', Freight: 32.38, Verified: !0
  },
  {
    OrderID: 10249, CustomerID: 'TOMSP', EmployeeID: 6, OrderDate: new Date("07 12 1996 00:03:23"),
    ShipName: 'Toms Spezialitäten', ShipCity: 'Münster', ShipAddress: 'Luisenstr. 48',
    ShipRegion: 'CJ', ShipPostalCode: '44087', ShipCountry: 'Germany', Freight: 11.61, Verified: !1
  }
]
```

```

},
{
  OrderID: 10250, CustomerID: 'HANAR', EmployeeID: 4, OrderDate: new Date("07 12 1996 00:00:23"),
  ShipName: 'Hanari Carnes', ShipCity: 'Rio de Janeiro', ShipAddress: 'Rua do Paço, 67',
  ShipRegion: 'RJ', ShipPostalCode: '05454-876', ShipCountry: 'Brazil', Freight: 65.83, Verified: !0
},
....
];
`

```

Ensure that the GraphQL server is properly configured and dependencies are installed to proceed with the next steps.

## Step 2: Schema Definition for GraphQL Server

In the context of GraphQL, a schema defines the structure of the data that clients can query from the server. It serves as a contract between the client and the server, outlining the types of data available, the operations that can be performed, and how the data is related.

When integrating GraphQL with the Syncfusion Grid, defining a schema involves specifying the types of data the Grid expects to receive from the GraphQL server, along with any additional parameters for operations like sorting, filtering, and paging.

Here's how you can define a schema for the Syncfusion Grid:

- **Define Types:** Create types representing the structure of data retrieved from GraphQL queries. Since the `GraphQLAdaptor` in Syncfusion extends from `UrlAdaptor`, it expects a JSON response with specific properties:
  - **result:** An array containing the data entities.
  - **count:** The total number of records.
  - **aggregates:** Contains total aggregate data(optional).

For example, if your Grid displays orders, you might define types for `ReturnType` and `Order`:

```

`
type ReturnType {
  result: [Order]
  count: Int
  aggregates: String # Total records aggregates
}

type Order {
  OrderID: Int!
  CustomerID: String

```

```

EmployeeID: Int
Freight: Int
ShipCity: String
ShipCountry: String
}
`

```

- **Define Queries:** Define queries that can be made to retrieve data from the server. In the case of a Grid, you may define a query to fetch orders, accepting parameters such as `DataManager` for advanced data operations. To utilize `Datamanager`, you need to install packages from `@syncfusion/ej2-data`

```

`
type Query {
getOrders(datamanager: DataManager): ReturnType
}
`

```

- **Define DataManager Input:** Define input types for `DataManager`, specifying parameters for sorting, filtering, paging, aggregates, etc., to be used in queries. The query parameters will be send in a string format which contains the below details.

Parameters	Description
-----	-----
<code>requiresCounts</code>	If it is <b>true</b> then the total count of records will be included in response.
<code>skip</code>	Holds the number of records to skip.
<code>take</code>	Holds the number of records to take.
<code>sorted</code>	Contains details about current sorted column and its direction.
<code>where</code>	Contains details about current filter column name and its constraints.
<code>group</code>	Contains details about current grouped column names.
<code>search</code>	Contains details about current search data.
<code>aggregates</code>	Contains details about aggregate data.

```

`
input DataManager {
skip: Int
take: Int

```

```

sorted: [Sort]
group: [String]
table: String
select: [String]
where: String
search: String
requiresCounts: Boolean
aggregates: [Aggregate]
params: String
}
`

```

Create a schema file (e.g., `src/schema.graphql`) in your GraphQL server project and write the schema definition there.

Grid Sort direction

```

input Sort {
  name: String!
  direction: String!
}

```

Grid aggregates type

```

input Aggregate {
  field: String!
  type: String!
}

```

Represents parameters for querying data, including sorting, filtering, etc.

```

input DataManager {
  skip: Int!
  take: Int!
  sorted: [Sort]
  group: [String]
  table: String
  select: [String]
  where: String
  search: String
}

```

```

requiresCounts: Boolean
aggregates: [Aggregate]
params: String
}

```

Represents an order type

```

type Order {
  OrderID: Int!
  CustomerID: String
  EmployeeID: Int
  Freight: Int
  ShipCity: String
  ShipCountry: String
}

```

Represents the result of a query, including the data and count

```

type ReturnType {
  result: [Order]
  count: Int
  aggregates: String
}

```

Represents a query to fetch orders with specified data manager parameters

```

type Query {
  getOrders(datamanager: DataManager): ReturnType
}
`

```

### Step 3: Implement Resolvers

To handle GraphQL queries and fetch data from your database, you need to create resolver functions. These resolver functions will be responsible for processing GraphQL queries and returning the appropriate data. In order to return data based on the grid expected result and count, utilize `DataUtil` from `@syncfusion/ej2-data` package.

Create a resolver file(**src/resolvers.js**) and implement the following code.

```

`javascript
import { OrderData } from "../db";
import { DataUtil } from "@syncfusion/ej2-data";
const resolvers = {
  Query: {

```

```
getOrders: (parent, { datamanager }, context, info) => {  
  var ret = DataUtil.processData(datamanager, OrderData);  
  return ret;  
}  
}  
};  
  
export default resolvers;  
`
```

#### Step 4: Run GraphQL Server

Install required packages and start the GraphQL server by running the following commands in your terminal:

```
`bash  
npm install  
npm run dev  
`
```

The server will be hosted at **http://localhost:xxxx/**. (where xxxx represents the port number).

#### [Connecting grid to an GraphQL service](#)

To integrate GraphQL with the Syncfusion Grid in your Angular application, follow the below steps:

#### Step 1: Create an Syncfusion Angular Grid:

Start a new Angular application using below Angular CLI command.

```
`bash  
ng new GridClient  
`
```

This command will prompt you for a few settings for the new project, such as whether to add Angular routing and which stylesheet format to use.

```
`bash  
cd GridClient  
`
```

#### Step 2: Adding Syncfusion Grid Package

To use Syncfusion Grid component and Datamanager, install the packages using the below command.

```
`bash  
npm i @syncfusion/ej2-data  
npm install @syncfusion/ej2-angular-grids --save  
`
```

**Step 3: Registering Grid Module**

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

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

**Step 4: 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
[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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-notifications/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

**Step 4: Configure DataManager with GraphQLAdaptor:**

Set up the `DataManager` with the `GraphQLAdaptor` to communicate with the GraphQL server. Then define the GraphQL query to specify the expected data structure and retrieval parameters:

```
`ts
[app.component.ts]
import { Component, OnInit } from '@angular/core';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90 isPrimaryKey='true'></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
    <e-column field='EmployeeID' headerText='Employee ID' textAlign='Right' width=90></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width=120></e-column>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: DataManager;
  ngOnInit(): void {
    // Configure DataManager with GraphQLAdaptor
    this.data = new DataManager({
      url: "http://localhost:xxxx/", // xxxx represents the port number
      adaptor: new GraphQLAdaptor({
        response: {
          result: 'getOrders.result', // Retrieve the actual order data
          count: 'getOrders.count' // Retrieve the total count of orders
        },
        // GraphQL query to fetch orders
        query: `query getOrders($datamanager: DataManager) {
          getOrders(datamanager: $datamanager) {
            count,
            result{`
```



```

OrderID, CustomerID, EmployeeID, ShipCountry}
}
},
})
});
}
}
,

```

### Step 5: Run the Application:

Once the GraphQL server is running, assign its URL (e.g., `http://localhost:xxxx/`) to the `dataManager.url` property of the `DataManager` in your Angular application.

```

`bash
npm i
ng serve
,

```

By following these steps, you will successfully integrate GraphQL with the Syncfusion Grid in your Angular application. Ensure that the GraphQL server is running smoothly and is accessible at the specified URL.

You can find the complete `GraphQLAdaptor` sample in the [GitHub](#) link.

### Handling searching operation

To handle search operation in the Syncfusion Grid using the `GraphQLAdaptor`, by utilizing the `datamanager.search` parameters and executing the search operation with the `search` method. This feature allows users to efficiently search through the grid's data and retrieve relevant information based on specified criteria.

In the image below, you can see the values of `datamanager.search` parameters:



### RESOLVER.JS

```

import { OrderData } from "../db";
import { DataUtil, Query, DataManager } from "@syncfusion/ej2-data";
const resolvers = {
  Query: {
    getOrders: (parent, { datamanager }, context, info) => {

```

```

let orders = [...OrderData];
const query = new Query();
const performSearching = (searchParam) => {
const { fields, key } = JSON.parse(searchParam)[0];
query.search(key, fields);
}
// Perform Searching
if (datamanager.search) {
performSearching(datamanager.search);
}
orders = new DataManager(orders).executeLocal(query);
var count = orders.length;
return { result: orders, count: count }; // Return result and count
},
},
};
export default resolvers;

```

### APP.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
selector: 'app-root',
template: `<ejs-grid [dataSource]='data' [toolbar]="toolbar">
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90
isPrimaryKey=true></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipCity' headerText='Ship City' textAlign='Right'
width=90></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=120></e-
column>
</e-columns>
</ejs-grid>
`,
styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
public data!: DataManager;
public toolbar?: string[];
ngOnInit(): void {
this.data = new DataManager({
url: "http://localhost:xxxx/", // Here xxxx denotes port number
adaptor: new GraphQLAdaptor({
response: {
result: 'getOrders.result',
count: 'getOrders.count'
},
query: `query getOrders($datamanager: DataManager) {
getOrders(datamanager: $datamanager) {
count,
result{
OrderID, CustomerID, ShipCity, ShipCountry}
}
} `
},

```

```

    })
  });
  this.toolbar = ['Search'];
}
}

```

## APP.MODULE.TS

```

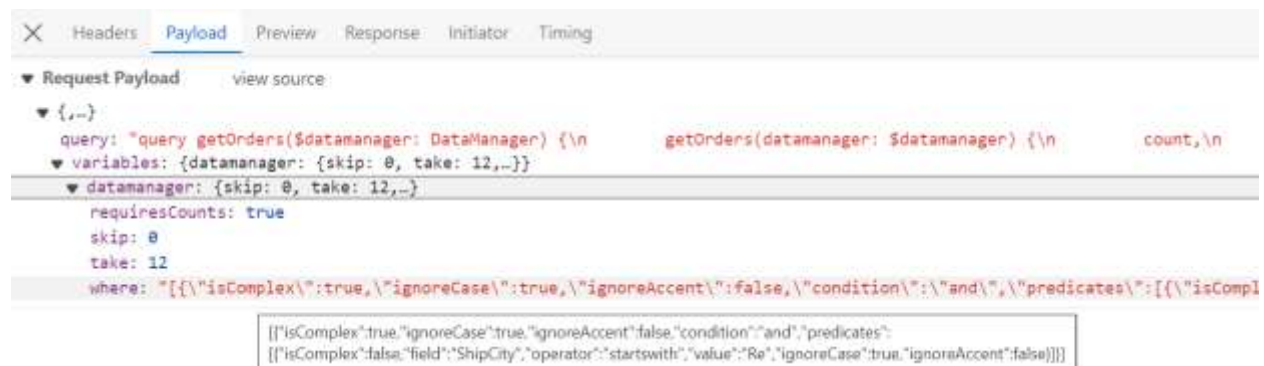
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the GridModule for the Grid component
import { GridModule, ToolbarService } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-grids module into NgModule
  imports: [BrowserModule, GridModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [ToolbarService]
})
export class AppModule { }

```

### Handling filtering operation

To handle filter operation in the Syncfusion Grid using the GraphQLAdaptor, by utilizing the `datamanager.where` parameters and executing the filter operation with the `where` method. This feature allows you to efficiently filter through the grid's data and retrieve relevant information based on specified criteria.

In the image below, you can see the values of `datamanager.where` parameters:



## RESOLVER.JS

```

import { OrderData } from './db';
import { DataUtil, Query, DataManager } from '@syncfusion/ej2-data';
const resolvers = {
  Query: {
    getOrders: (parent, { datamanager }, context, info) => {
      let orders = [...OrderData];
      const query = new Query();
      const performFiltering = (filterString) => {
        const filter = JSON.parse(filterString);
        // Iterating over each predicate
        filter[0].predicates.forEach(predicate => {

```

```

const field = predicate.field;
const operator = predicate.operator;
const value = predicate.value;
query.where(field, operator, value);
});
}
// Perform filtering
if (datamanager.where) {
  performFiltering(datamanager.where);
}
orders = new DataManager(orders).executeLocal(query);
var count = orders.length;
return { result: orders, count: count }; // Return result and count
},
},
};
export default resolvers;

```

### APP.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `
<ejs-grid [dataSource]='data' [allowFiltering]="true">
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90
isPrimaryKey=true></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipCity' headerText='Ship City' textAlign='Right'
width=90></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=120></e-
column>
</e-columns>
</ejs-grid>
`,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  public data!: DataManager;
  ngOnInit(): void {
    this.data = new DataManager({
      url: "http://localhost:xxxx/", // Here xxxx denotes port number
      adaptor: new GraphQLAdaptor({
        response: {
          result: 'getOrders.result',
          count: 'getOrders.count'
        },
        query: `query getOrders($datamanager: DataManager) {
          getOrders(datamanager: $datamanager) {
            count,
            result{
              OrderID, CustomerID, ShipCity, ShipCountry
            }
          }
        }`
      })
    });
  }
}

```

```

})
});
}
}

```

## APP.MODULE.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the GridModule for the Grid component
import { FilterService, GridModule } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from '../app.component';
@NgModule({
  //declaration of ej2-angular-grids module into NgModule
  imports: [BrowserModule, GridModule],
  declarations: [AppComponent],
  bootstrap: [AppComponent],
  providers: [FilterService]
})
export class AppModule { }

```

### Handling sorting operation

To handle sort operation in the Syncfusion Grid using the GraphQLAdaptor, by utilizing the `datamanager.sorted` parameters and executing the sort operation with the [sortBy](#) method. This feature allows users to efficiently sort grid data based on specified criteria.

In the image below, you can see the values of `datamanager.sorted` parameters:



## RESOLVER.JS

```

import { OrderData } from '../db';
import { DataUtil, Query, DataManager } from '@syncfusion/ej2-data';
const resolvers = {
  Query: {
    getOrders: (parent, { datamanager }, context, info) => {
      let orders = [...OrderData];
      const query = new Query();
      const performSorting = (sorted) => {
        for (let i = 0; i < sorted.length; i++) {
          const { name, direction } = sorted[i];
          query.sortBy(name, direction);
        }
      }
      // Perform sorting
    }
  }
}

```

```

if (datamanager.sorted) {
  performSorting(datamanager.sorted);
}
orders = new DataManager(orders).executeLocal(query);
var count = orders.length;
return { result: orders, count: count }; // Return result and count
},
},
};
export default resolvers;

```

## APP.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `
    <ejs-grid [dataSource]='data' [allowSorting]="true">
    <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90
    isPrimaryKey='true'></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
    <e-column field='ShipCity' headerText='Ship City' textAlign='Right'
    width=90></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width=120></e-
    column>
    </e-columns>
    </ejs-grid>
    `
  ,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  public data!: DataManager;
  public pageSettings!: PageSettingsModel;
  ngOnInit(): void {
    this.data = new DataManager({
      url: "http://localhost:xxxx/", // Here xxxx denotes port number
      adaptor: new GraphQLAdaptor({
        response: {
          result: 'getOrders.result',
          count: 'getOrders.count'
        },
        query: `query getOrders($datamanager: DataManager) {
          getOrders(datamanager: $datamanager) {
            count,
            result{
              OrderID, CustomerID, ShipCity, ShipCountry
            }
          }
        }`,
      })
    });
  }
}

```

**APP.MODULE.TS**

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

*Handling paging operation*

To handle page operation in the Syncfusion Grid using the GraphQLAdaptor, by utilizing the `datamanager.skip` and `datamanager.take` parameters and executing the paging with the [page](#) method. This feature allows users to navigate through large datasets efficiently by dividing them into pages.

In the image below, you can see the value of `datamanager.skip` and `datamanager.take` parameters:

**RESOLVER.JS**

```
import { OrderData } from './db';
import { DataUtil, Query, DataManager } from '@syncfusion/ej2-data';
const resolvers = {
  Query: {
    getOrders: (parent, { datamanager }, context, info) => {
      let orders = [...OrderData];
      const query = new Query();
      // Perform paging
      if (datamanager.skip && datamanager.take) {
        const pageSkip = datamanager.skip / datamanager.take + 1;
        const pageTake = datamanager.take;
        query.page(pageSkip, pageTake);
      } else if (datamanager.skip === 0 && datamanager.take) {
        query.page(1, datamanager.take);
      }
      const currentResult = new DataManager(orders).executeLocal(query);
      return { result: currentResult, count: count }; // Return result and count
    },
  },
};
export default resolvers;
```

**APP.COMPONENT.TS**

```

import { Component, OnInit } from '@angular/core';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `
<ejs-grid [dataSource]='data' [allowPaging]="true"
[pageSettings]="pageSettings" >
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90
isPrimaryKey='true'></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipCity' headerText='Ship City' textAlign='Right'
width=90></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=120></e-
column>
</e-columns>
</ejs-grid>
`,
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  public data!: DataManager;
  public pageSettings!: PageSettingsModel;
  ngOnInit(): void {
    this.data = new DataManager({
      url: "http://localhost:xxxx/", // Here xxxx denotes port number
      adaptor: new GraphQLAdaptor({
        response: {
          result: 'getOrders.result',
          count: 'getOrders.count'
        },
        query: `query getOrders($datamanager: DataManager) {
          getOrders(datamanager: $datamanager) {
            count,
            result{
              OrderID, CustomerID, ShipCity, ShipCountry
            }
          }
        }`,
      })
    });
    this.pageSettings = { pageSize: 12 };
  }
}

```

**APP.MODULE.TS**

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the GridModule for the Grid component
import { PageService } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-grids module into NgModule

```



```
imports: [BrowserModule, GridModule],
declarations: [AppComponent],
bootstrap: [AppComponent],
providers: [PageService]
}))
export class AppModule { }
```

### Handling CRUD operations

Syncfusion Grid seamlessly integrates with GraphQL servers using the **GraphQLAdaptor**, enabling efficient CRUD (Create, Read, Update, Delete) operations on your data. The below steps explain how to perform CRUD actions using **GraphQLAdaptor** in Syncfusion Grid.

#### Insert operation

Adding a new record to the database involves the following steps:

- Define the **createOrder** mutation in your GraphQL schema to handle creating a new order. This mutation should accept an **OrderInput** object containing the new order details.

,

```
[schema.graphql]
```

```
input OrderInput {
```

```
  OrderID: Int!
```

```
  CustomerID: String
```

```
  Freight: Float
```

```
  ShipCity: String
```

```
  ShipCountry: String
```

```
}
```

```
type Mutation {
```

```
  createOrder(value: OrderInput): Order!
```

```
}
```

,

- Implement the **createOrder** resolver function in your resolver file. This function should add the new order data to your data source and return the newly created order object.

```
`js
```

```
[resolver.js]
```

```
Mutation: {
```

```
  createOrder: (parent, { value }, context, info) => {
```

```
    const newOrder = value;
```

```

OrderData.push(newOrder);
return newOrder;
},
}
`

```

- Configure the **getMutation** function in your **GraphQLAdaptor** to return the appropriate GraphQL mutation query string based on the insert action. This query string should reference the **createOrder** mutation defined in your schema.

```

`ts
[app.component.ts]
// mutation for performing insert operation
getMutation: function (action: string) {
  if (action === 'insert') {
    return `mutation CreateOrderMutation($value: OrderInput!){
      createOrder(value: $value){
        OrderID, CustomerID, Freight, ShipCity, ShipCountry
      }
    }`;
  }
}
`

```

### Update Operation

Updating an existing record in the database involves the following steps:

- Define the **updateOrder** mutation in your GraphQL schema to handle updating an order. This mutation should accept three arguments:
- **key**: The unique identifier of the order to be updated.
- **keyColumn**: The name of the column containing the unique identifier.
- **value**: An OrderInput object containing the updated order details.

```

[schema.graphql]
type Order {
  OrderID: Int!
  CustomerID: String
  Freight: Float

```

```

ShipCity: String
ShipCountry: String
}
input OrderInput {
  OrderID: Int!
  CustomerID: String
  Freight: Float
  ShipCity: String
  ShipCountry: String
}
type Mutation {
  updateOrder(key: Int!, keyColumn: String, value: OrderInput): Order
}
`

```

- Implement the **updateOrder** resolver function in your resolver file. This function should find the order based on the provided key and keyColumn, update its properties with the values from the value argument, and return the updated order object.

```

`js
[resolver.js]
Mutation: {
  updateOrder: (parent, { key, keyColumn, value }, context, info) => {
    let updatedOrder = OrderData.find(order => order.OrderID === parseInt(key));
    updatedOrder.CustomerID = value.CustomerID;
    updatedOrder.EmployeeID = value.EmployeeID;
    updatedOrder.Freight = value.Freight;
    updatedOrder.ShipCity = value.ShipCity;
    updatedOrder.ShipCountry = value.ShipCountry;
    return updatedOrder; // Make sure to return the updated order.
  },
}
`

```

- Configure the **getMutation** function in your GraphQLAdaptor to return the appropriate GraphQL mutation query string based on the update action. This query string should reference the **updateOrder** mutation defined in your schema.

```
`ts
[app.component.ts]
// mutation for performing update operation
getMutation: function (action: any): string {
  if (action === 'update') {
    return `mutation UpdateOrderMutation($key: Int!, $keyColumn: String,$value: OrderInput){
      updateOrder(key: $key, keyColumn: $keyColumn, value: $value) {
        OrderID, CustomerID, Freight, ShipCity, ShipCountry
      }
    }`;
  }
}
```

### Delete Operation

Deleting a record from the database involves the following steps:

- Define the **deleteOrder** mutation in your GraphQL schema to handle deleting an order. This mutation should accept three arguments similar to the updateOrder mutation.

```
[schema.graphql]
type Order {
  OrderID: Int!
  CustomerID: String
  Freight: Float
  ShipCity: String
  ShipCountry: String
}
input OrderInput {
  OrderID: Int!
  CustomerID: String
  Freight: Float
  ShipCity: String
```

```
ShipCountry: String
```

```
}
```

```
type Mutation {
```

```
  deleteOrder(key: Int!, keyColumn: String, value: OrderInput): Order!
```

```
}
```

```
,
```

- Implement the **deleteOrder** resolver function in your resolver file. This function should find the order based on the provided key and keyColumn, remove it from your data source, and return the deleted order object.

```
`js
```

```
[resolver.js]
```

```
Mutation: {
```

```
  deleteOrder: (parent, { key, keyColumn, value }, context, info) => {
```

```
    const orderIndex = OrderData.findIndex(order => order.OrderID === parseInt(key));
```

```
    if (orderIndex === -1) throw new Error("Order not found." + value);
```

```
    const deletedOrders = OrderData.splice(orderIndex, 1);
```

```
    return deletedOrders[0];
```

```
}
```

```
}
```

```
,
```

- Configure the **getMutation** function in your GraphQL adaptor to return the appropriate GraphQL mutation query string based on the delete action. This query string should reference the **deleteOrder** mutation defined in your schema.

```
.
```

```
`ts
```

```
[app.component.ts]
```

```
// mutation for performing delete operation
```

```
getMutation: function (action: string) {
```

```
  if (action === 'delete') {
```

```
    return `mutation RemoveOrderMutation($key: Int!, $keyColumn: String, $value: OrderInput){
```

```
      deleteOrder(key: $key, keyColumn: $keyColumn, value: $value) {
```

```
        OrderID, CustomerID, Freight, ShipCity, ShipCountry
```

```
      }
    }
  }
};
```

```

}
}
,

```

Normal/Inline editing is the default edit [mode](#) for the Grid component. To enable CRUD operations, ensure that the [isPrimaryKey](#) property is set to **true** for a specific grid column, ensuring that its value is unique.

### **RESOLVER.JS**

```

import { OrderData } from "../db";
import { DataUtil } from "@syncfusion/ej2-data";
const resolvers = {
  Query: {
    getOrders: (parent, { datamanager }, context, info) => {
      var ret = DataUtil.processData(datamanager, OrderData);
      return ret;
    }
  },
  Mutation: {
    createOrder: (parent, { value }, context, info) => {
      const newOrder = value;
      OrderData.push(newOrder);
      return newOrder;
    },
    updateOrder: (parent, { key, keyColumn, value }, context, info) => {
      let updatedOrder = OrderData.find(order => order.OrderID === parseInt(key));
      updatedOrder.CustomerID = value.CustomerID;
      updatedOrder.EmployeeID = value.EmployeeID;
      updatedOrder.Freight = value.Freight;
      updatedOrder.ShipCity = value.ShipCity;
      updatedOrder.ShipCountry = value.ShipCountry;
      return updatedOrder; // Make sure to return the updated order.
    },
    deleteOrder: (parent, { key, keyColumn, value }, context, info) => {
      const orderIndex = OrderData.findIndex(order => order.OrderID ===
        parseInt(key));
      if (orderIndex === -1) throw new Error("Order not found." + value);
      const deletedOrders = OrderData.splice(orderIndex, 1);
      return deletedOrders[0];
    }
  }
};
export default resolvers;

```

### **SCHEMA.GRAPHQL**

```

#Grid Sort direction
input Sort {
  name: String
  direction: String
}
#Grid aggregates type
input Aggregate {
  field: String!
}

```

```

type: String!
}
#Syncfusion DataManager query params
input DataManager {
  skip: Int
  take: Int
  sorted: [Sort]
group: [String]
  table: String
select: [String]
where: String
  search: String
  requiresCounts: Boolean,
  aggregates: [Aggregate],
params: String
}
#Grid field names
input OrderInput {
  OrderID: Int!
  CustomerID: String
  ShipCity: String
  ShipCountry: String
}
type Order {
  OrderID: Int!
  CustomerID: String
  ShipCity: String
  ShipCountry: String
}
#need to return type as 'result (i.e, current pager data)' and count (i.e.,
total number of records in your database)
type ReturnType {
  result: [Order]
  count: Int
  aggregates: String
}
type Query {
  getOrders(datamanager: DataManager): ReturnType
}
type Mutation {
  createOrder(value: OrderInput): Order!
  updateOrder(key: Int!, keyColumn: String, value: OrderInput): Order
  deleteOrder(key: Int!, keyColumn: String, value: OrderInput): Order!
}

```

### APP.COMPONENT.TS

```

import { Component, OnInit } from '@angular/core';
import { EditSettingsModel } from '@syncfusion/ej2-angular-grids';
import { DataManager, GraphQLAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `
<ejs-grid [dataSource]='data' [editSettings]="editSettings"
[toolbar]="toolbar">
<e-columns>

```

```

<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90
isPrimaryKey='true'></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipCity' headerText='Ship City' textAlign='Right'
width=90></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=120></e-
column>
</e-columns>
</ejs-grid>
`,
styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
public data!: DataManager;
public pageSettings!: PageSettingsModel;
public editSettings?: EditSettingsModel;
public toolbar?: string[];
ngOnInit(): void {
this.data = new DataManager({
url: "http://localhost:xxxx/", // Here xxxx denotes port number
adaptor: new GraphQLAdaptor({
response: {
result: 'getOrders.result',
count: 'getOrders.count'
},
query: `query getOrders($datamanager: DataManager) {
getOrders(datamanager: $datamanager) {
count,
result{
OrderID, CustomerID, ShipCity, ShipCountry}
}
} `,
// mutation for performing CRUD
getMutation: function (action: string) {
if (action === 'insert') {
return `mutation CreateOrderMutation($value: OrderInput!){
createOrder(value: $value){
OrderID, CustomerID, ShipCity, ShipCountry
}}`;
}
if (action === 'update') {
return `mutation UpdateOrderMutation($key: Int!, $keyColumn: String,$value:
OrderInput){
updateOrder(key: $key, keyColumn: $keyColumn, value: $value) {
OrderID, CustomerID, ShipCity, ShipCountry
}
} `;
} else {
return `mutation RemoveOrderMutation($key: Int!, $keyColumn: String, $value:
OrderInput){
deleteOrder(key: $key, keyColumn: $keyColumn, value: $value) {
OrderID, CustomerID, ShipCity, ShipCountry
}
} `;
}
}
})

```



```
});  
this.editSettings = { allowAdding: true, allowDeleting: true, allowEditing:  
true, mode: 'Normal' };  
this.toolbar = ['Add', 'Delete', 'Update', 'Cancel'];  
}  
}
```

### **APP.MODULE.TS**

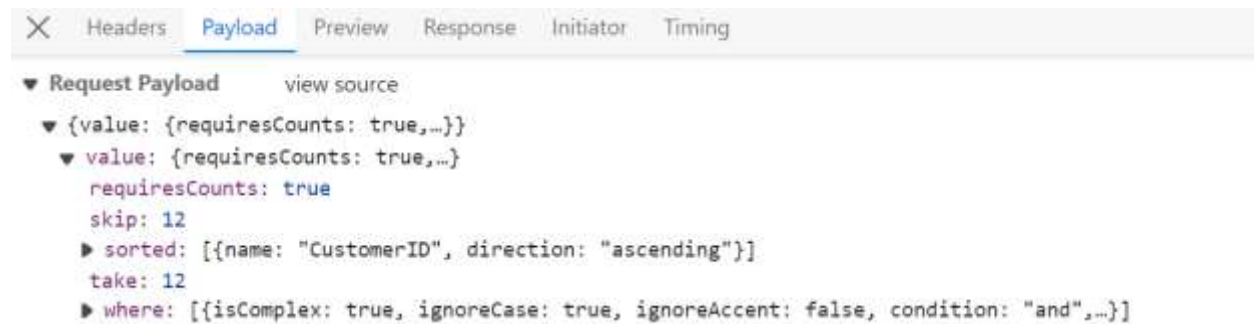
```
import { NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
// import the GridModule for the Grid component  
import { EditService, ToolbarService } from '@syncfusion/ej2-angular-grids';  
import { AppComponent } from './app.component';  
@NgModule({  
  //declaration of ej2-angular-grids module into NgModule  
  imports: [BrowserModule, GridModule],  
  declarations: [AppComponent],  
  bootstrap: [AppComponent],  
  providers: [ EditService, ToolbarService]  
})  
export class AppModule { }
```

+ Add    🗑 Delete    📄 Update    ✕ Cancel				
Order ID	Customer ID	Freight	Ship City	Ship Country
10248	VINET	32.38	Reims	France
10249	TOMSP	11.61	Münster	Germany
10250	VINET	65.83	Rio de Jan...	France
10251	VICTE	41.34	Lyon	France
10253	HANAR	58.17	Rio de Jan...	Brazil
10254	CHOPS	22.98	Bern	Switzerland
10255	RICSU	148.33	Genève	Switzerland
10256	WELLI	13.97	Resende	Brazil
10257	HILAA	81.91	San Cristó...	Venezuela
10258	ERNSH	140.51	Graz	Austria
10259	CENTC	3.25	México D.F.	Mexico
10260	OTTIK	55.09	Köln	Germany
<      <      1 of 8 pages      >      >				

You can find the complete GraphQLAdaptor sample in the [GitHub](#) link.

#### WebMethodAdaptor in Syncfusion Angular Grid Component

The [WebMethodAdaptor](#) in Syncfusion Angular Grid facilitates data binding from remote services using web methods. This powerful feature enables efficient communication between the client-side application and the server. The WebMethodAdaptor, like the URL adaptor, sends query parameters encapsulated within an object named **value**. Within this **value** object, various datamanager properties such as **requiresCounts**, **skip**, **take**, **sorted**, and **where** queries are included.



This section describes a step-by-step process for retrieving data using WebMethodAdaptor, then binding it to the Angular Grid component to facilitate data and CRUD operations.

### *Creating an API service*

To configure a server with Syncfusion Angular Grid, you need to follow the below steps:

#### **1. Project Creation:**

Open Visual Studio and create an Angular and ASP.NET Core project named **WebMethodAdaptor**. To create an Angular and ASP.NET Core application, follow the documentation [link](#) for detailed steps.

#### **2. Model Class Creation:**

Create a model class named **OrdersDetails.cs** in the server-side **Models** folder to represent the order data.

#### **ORDERSDETAILS.CS**

```
namespace WebMethodAdaptor.Server.Models
{
    public class OrdersDetails
    {
        public static List<OrdersDetails> order = new List<OrdersDetails>();
        public OrdersDetails()
        {
        }
        public OrdersDetails(
            int OrderID, string CustomerId, int EmployeeId, double Freight, bool
            Verified,
            DateTime OrderDate, string ShipCity, string ShipName, string ShipCountry,
            DateTime ShippedDate, string ShipAddress)
        {
            this.OrderID = OrderID;
            this.CustomerID = CustomerId;
            this.EmployeeID = EmployeeId;
            this.Freight = Freight;
            this.ShipCity = ShipCity;
            this.Verified = Verified;
            this.OrderDate = OrderDate;
            this.ShipName = ShipName;
            this.ShipCountry = ShipCountry;
            this.ShippedDate = ShippedDate;
            this.ShipAddress = ShipAddress;
        }
        public static List<OrdersDetails> GetAllRecords()
        {
            if (order.Count() == 0)
            {
            }
        }
    }
}
```

```

{
    int code = 10000;
    for (int i = 1; i < 10; i++)
    {
        order.Add(new OrdersDetails(code + 1, "ALFKI", i + 0, 2.3 * i, false, new
        DateTime(1991, 05, 15), "Berlin", "Simons bistro", "Denmark", new
        DateTime(1996, 7, 16), "Kirchgasse 6"));
        order.Add(new OrdersDetails(code + 2, "ANATR", i + 2, 3.3 * i, true, new
        DateTime(1990, 04, 04), "Madrid", "Queen Cozinha", "Brazil", new
        DateTime(1996, 9, 11), "Avda. Azteca 123"));
        order.Add(new OrdersDetails(code + 3, "ANTON", i + 1, 4.3 * i, true, new
        DateTime(1957, 11, 30), "Cholchester", "Frankenversand", "Germany", new
        DateTime(1996, 10, 7), "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"));
        order.Add(new OrdersDetails(code + 4, "BLONP", i + 3, 5.3 * i, false, new
        DateTime(1930, 10, 22), "Marseille", "Ernst Handel", "Austria", new
        DateTime(1996, 12, 30), "Magazinweg 7"));
        order.Add(new OrdersDetails(code + 5, "BOLID", i + 4, 6.3 * i, true, new
        DateTime(1953, 02, 18), "Tsawassen", "Hanari Carnes", "Switzerland", new
        DateTime(1997, 12, 3), "1029 - 12th Ave. S.));
        code += 5;
    }
}
return order;
}

public int? OrderID { get; set; }
public string? CustomerID { get; set; }
public int? EmployeeID { get; set; }
public double? Freight { get; set; }
public string? ShipCity { get; set; }
public bool? Verified { get; set; }
public DateTime OrderDate { get; set; }
public string? ShipName { get; set; }
public string? ShipCountry { get; set; }
public DateTime ShippedDate { get; set; }
public string? ShipAddress { get; set; }
}
}

```

### 3. API Controller Creation:

Create a file named `GridController.cs` under the **Controllers** folder. This controller will handle data communication with the Angular Grid component.

#### GRIDCONTROLLER.CS

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using WebMethodAdaptor.Server.Models;
namespace WebMethodAdaptor.Server.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class GridController : ControllerBase
    {
        // method to retrieve data
        [HttpGet]

```

```
[Route("api/[controller]")]
public List<OrdersDetails> GetOrderData()
{
    // Retrieve all records and convert to list
    var data = OrdersDetails.GetAllRecords().ToList();
    return data;
}
// method to handle incoming data manager requests
[HttpPost]
[Route("api/[controller]")]
public object Post()
{
    // Retrieve data source and convert to queryable
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    // Initialize QueryableOperation
    QueryableOperation queryableOperation = new QueryableOperation();
    // Get total record count
    int totalRecordsCount = DataSource.Count();
    // Return result and total record count
    return new { result = DataSource, count = totalRecordsCount };
}
}
```

The **GetOrderData** method retrieves sample order data. You can replace it with your custom logic to fetch data from a database or any other source.

#### 4. Run the Application:

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

After running the application, you can verify that the server-side API controller is successfully returning the order data in the URL(<https://localhost:xxxx/api/Grid>). Here **xxxx** denotes the port number.



```

1  [
2    {
3      "orderID": 10001,
4      "customerID": "ALFKI",
5      "employeeID": 1,
6      "freight": 2.3,
7      "shipCity": "Berlin",
8      "verified": false,
9      "orderDate": "1991-05-15T00:00:00",
10     "shipName": "Simons bistro",
11     "shipCountry": "Denmark",
12     "shippedDate": "1996-07-16T00:00:00",
13     "shipAddress": "Kirchgasse 6"
14   },
15   {
16     "orderID": 10002,
17     "customerID": "ANATR",
18     "employeeID": 3,
19     "freight": 3.3,
20     "shipCity": "Madrid",
21     "verified": true,
22     "orderDate": "1990-04-04T00:00:00",
23     "shipName": "Queen Cozinha",
24     "shipCountry": "Brazil",
25     "shippedDate": "1996-09-11T00:00:00",
26     "shipAddress": "Avda. Azteca 123"
27   },
28   {
29     "orderID": 10003,
30     "customerID": "ANTON",
31     "employeeID": 2,
32     "freight": 4.3,
33     "shipCity": "Cholchester",
34     "verified": true,
35     "orderDate": "1957-11-30T00:00:00",
36     "shipName": "Frankenversand",
37     "shipCountry": "Germany",
38     "shippedDate": "1996-10-07T00:00:00",
39     "shipAddress": "Carrera 52 con Ave. Bolívar #65-98 Llano Largo"
40   },
41   {
42     "orderID": 10004,
43     "customerID": "BLONP",
44     "employeeID": 4,
45     "freight": 5.3,

```

### [Connecting Syncfusion Angular Grid to an API service](#)

To integrate the Syncfusion Grid component into your Angular and ASP.NET Core project using Visual Studio, follow the below steps:

#### 1: Install Syncfusion Package

Open your terminal in the project's client folder and install the required Syncfusion packages using npm:

```
`bash
```

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

```
npm install @syncfusion/ej2-data --save
```

## 2: Import Grid Module

In the `app.module.ts` file, import the **GridModule** from the `@syncfusion/ej2-angular-grids` package:

### APP.MODULE.TS

```
{% raw %}
import { GridModule } from '@syncfusion/ej2-angular-grids';
@NgModule({
  imports: [
    // Other imports...
    GridModule,
  ],
})
export class AppModule { }
{% endraw %}
```

## Step 3: Adding CSS reference

Include the necessary CSS files in your `styles.css` file to style the Syncfusion Angular component:

### 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-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-splitbuttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-grids/styles/material.css';
```

## Step 4: Adding Syncfusion Component

In your component file (e.g., `app.component.ts`), import **DataManager** and **WebMethodAdaptor** from `@syncfusion/ej2-data`. Create a **DataManager** instance specifying the URL of your API endpoint(`https://localhost:xxxx/api/Grid`) using the `url` property and set the adaptor **WebMethodAdaptor**.

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = new DataManager({
```

```
url: 'https://localhost:xxxx/api/grid', // Here xxxx represents the port
    number
    adaptor: new WebMethodAdaptor()
  });
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' width='120'
      textAlign='Right' isPrimaryKey="true"></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width='160'></e-
      column>
    <e-column field='ShipCity' headerText='Ship City' width='150'></e-column>
    <e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
      column>
  </e-columns>
</ejs-grid>
```

Replace `https://localhost:xxxx/api/grid` with the actual **URL** of your API endpoint that provides the data in a consumable format (e.g., JSON).

Run the application in Visual Studio. It will be accessible on a URL like **https://localhost:xxxx**.

Ensure your API service is configured to handle CORS (Cross-Origin Resource Sharing) if necessary.

`cs

[program.cs]

```
builder.Services.AddCors(options =>
{
  options.AddDefaultPolicy(builder =>
  {
    builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
  });
});

var app = builder.Build();
app.UseCors();
`
```



Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BLONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BLONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil
10013	ANTON	Cholchester	Germany

\* The Syncfusion Grid component provides built-in support for handling various data operations such as searching, sorting, filtering, aggregate and paging on the server-side. These operations can be handled using methods such as [PerformSearching](#), [PerformFiltering](#), [PerformSorting](#), [PerformTake](#) and [PerformSkip](#) available in the Syncfusion.EJ2.AspNet.Core package. Let's explore how to manage these data operations using the [WebMethodAdaptor](#).

\* In an API service project, add Syncfusion.EJ2.AspNet.Core by opening the NuGet package manager in Visual Studio (Tools → NuGet Package Manager → Manage NuGet Packages for Solution), search and install it.

\* To access [DataManagerRequest](#) and [QueryableOperation](#), import Syncfusion.EJ2.Base in [GridController.cs](#) file.

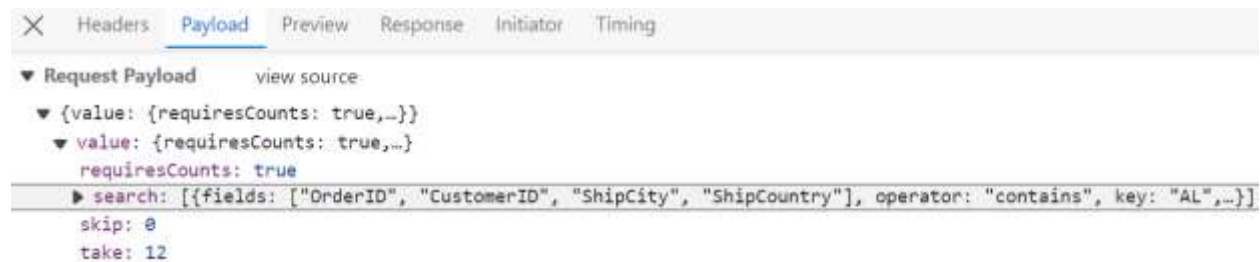
\* In the WebMethodAdaptor configuration, the properties of the DataManager object are encapsulated within an object named **value**. To access the DataManager properties, a dedicated class is created, representing the **value** object.

```
`cs
// Model for handling data manager requests
public class DataManager
{
    public required DataManagerRequest Value { get; set; }
```

```
}
,
```

### Handling searching operation

To handle searching operation, ensure that your API endpoint supports custom searching criteria. Implement the searching logic on the server-side using the [PerformSearching](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo searching based on the criteria specified in the incoming [DataManagerRequest](#) object



### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManager DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    // Initialize QueryableOperation instance
    // Retrieve data manager value
    DataManagerRequest DataManagerParams = DataManagerRequest.Value;
    // Handling Searching
    if (DataManagerParams.Search != null && DataManagerParams.Search.Count > 0)
    {
        DataSource = queryableOperation.PerformSearching(DataSource,
        DataManagerParams.Search);
    }
    // Get the total records count
    int totalRecordsCount = DataSource.Count();
    // Return data based on the request
    return new { result = DataSource, count = totalRecordsCount };
}

// Model for handling data manager requests
public class DataManager
{
    public required DataManagerRequest Value { get; set; }
}
```

### APP.COMPONENT.TS

```
import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
})
```

```
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
      adaptor: new WebMethodAdaptor()
    });
    this.toolbar = ['Search'];
  }
}
```

### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [toolbar]="toolbar" height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
      column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
  </e-columns>
</ejs-grid>
```

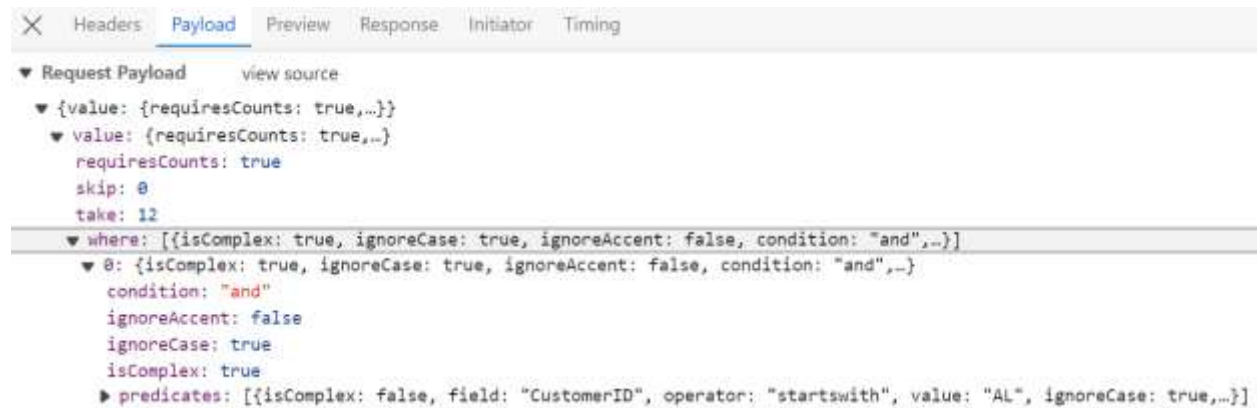
### APP.MODULE.TS

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, ToolbarService } from '@syncfusion/ej2-angular-grids';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, HttpClientModule,
    AppRoutingModuleModule, GridModule
  ],
  providers: [ToolbarService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

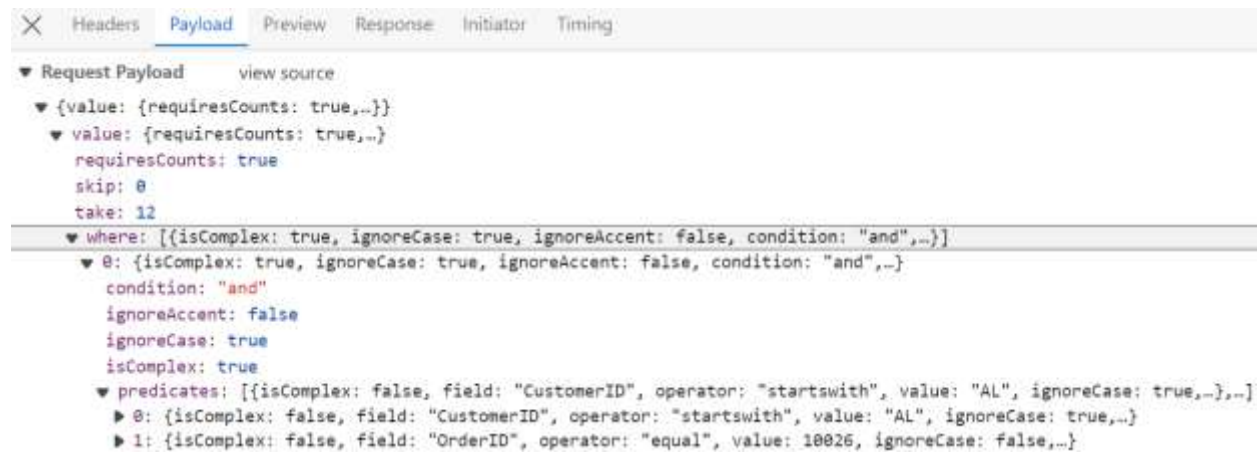
#### Handling filtering operation

To handle filtering operation, ensure that your API endpoint supports custom filtering criteria. Implement the filtering logic on the server-side using the [PerformFiltering](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo filtering based on the criteria specified in the incoming [DataManagerRequest](#) object.

## Single column filtering



## Multi column filtering



## GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManager DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Retrieve data manager value
    DataManagerRequest DataManagerParams = DataManagerRequest.Value;
    if (DataManagerParams.Where != null && DataManagerParams.Where.Count > 0)
    {
        // Handling filtering operation
        foreach (var condition in DataManagerParams.Where)
        {
            foreach (var predicate in condition.predicates)
            {
                DataSource = queryableOperation.PerformFiltering(DataSource,
                    DataManagerParams.Where, predicate.Operator);
            }
        }
    }
    // Get the total records count
```

```

int totalRecordsCount = DataSource.Count();
// Return data based on the request
return new { result = DataSource, count = totalRecordsCount };
}
// Model for handling data manager requests
public class DataManager
{
    public required DataManagerRequest Value { get; set; }
}

```

### APP.COMPONENT.TS

```

import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
})
export class AppComponent {
    public data?: DataManager;
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = new DataManager({
            url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
            adaptor: new WebMethodAdaptor()
        });
    }
}

```

### APP.COMPONENT.HTML

```

<ejs-grid #grid [dataSource]='data' allowFiltering='true' height="320">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
      width='150'></e-column>
    <e-column field='CustomerID' headerText='Customer Name' width='150'></e-
      column>
    <e-column field='ShipCity' headerText='ShipCity' width='150'
      textAlign='Right'></e-column>
  </e-columns>
</ejs-grid>

```

### APP.MODULE.TS

```

import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, FilterService } from '@syncfusion/ej2-angular-grids';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        BrowserModule,
        AppRoutingModuleModule,
        GridModule
    ],
    providers: [
        FilterService
    ],
    bootstrap: [AppComponent]
})
export class AppModule {}

```

```

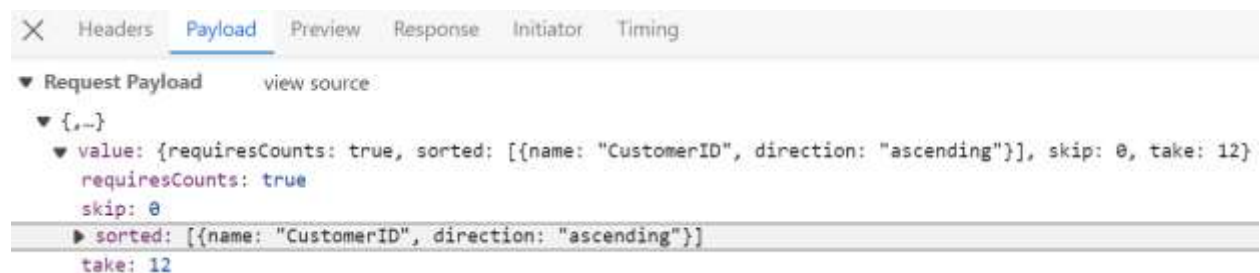
],
imports: [
  BrowserModule, HttpClientModule,
  AppRoutingModule, GridModule
],
providers: [FilterService],
bootstrap: [AppComponent]
})
export class AppModule { }

```

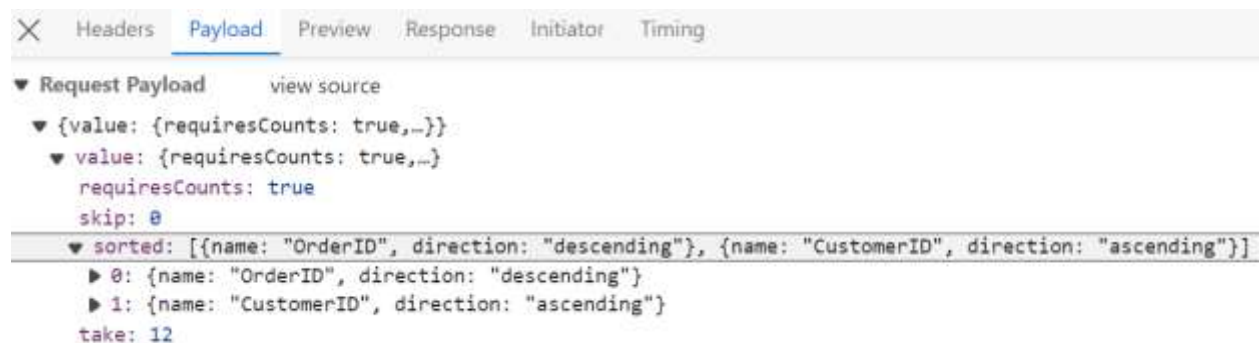
### Handling sorting operation

To handle sorting operation, ensure that your API endpoint supports custom sorting criteria. Implement the sorting logic on the server-side using the [PerformSorting](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo sorting based on the criteria specified in the incoming [DataManagerRequest](#) object.

### Single column sorting



### Multi column sorting



### GRIDCONTROLLER.CS

```

[HttpPost]
public object Post([FromBody] DataManager DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Retrieve data manager value
    DataManagerRequest DataManagerParams = DataManagerRequest.Value;
    // Handling Sorting operation
    if (DataManagerParams.Sorted != null && DataManagerParams.Sorted.Count > 0)
    {

```

```

DataSource = queryableOperation.PerformSorting(DataSource,
DataManagerParams.Sorted);
}
// Get the total count of records
int totalRecordsCount = DataSource.Count();
// Return data based on the request
return new { result = DataSource, count = totalRecordsCount };
}
// Model for handling data manager requests
public class DataManager
{
public required DataManagerRequest Value { get; set; }
}

```

### APP.COMPONENT.TS

```

import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
public data?: DataManager;
@ViewChild('grid')
public grid?: GridComponent;
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
adaptor: new WebMethodAdaptor()
});
}
}

```

### APP.COMPONENT.HTML

```

<ejs-grid #grid [dataSource]='data' allowSorting='true' height="320">
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='ShipCity' headerText='ShipCity' width='150'
textAlign='Right'></e-column>
</e-columns>
</ejs-grid>

```

### APP.MODULE.TS

```

import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids';
import { AppRoutingModuleModule } from './app-routing.module';

```

```
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule, HttpClientModule,
    AppRoutingModule, GridModule
  ],
  providers: [SortService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

### Handling paging operation

To handle paging operation, ensure that your API endpoint supports custom paging criteria. Implement the paging logic on the server-side using the [PerformTake](#) and [PerformSkip](#) method from the [QueryableOperation](#) class. This allows the custom data source to undergo paging based on the criteria specified in the incoming [DataManagerRequest](#) object.

✕ Headers **Payload** Preview Response Initiator Timing

▼ **Request Payload** [view source](#)

▼ {value: {requiresCounts: true, skip: 12, take: 12}}

▼ value: {requiresCounts: true, skip: 12, take: 12}

requiresCounts: true

skip: 12

take: 12

### GRIDCONTROLLER.CS

```
[HttpPost]
public object Post([FromBody] DataManager DataManagerRequest)
{
    // Retrieve data from the data source (e.g., database)
    IQueryable<OrdersDetails> DataSource = GetOrderData().AsQueryable();
    // Get the total records count
    int totalRecordsCount = DataSource.Count();
    QueryableOperation queryableOperation = new QueryableOperation(); //
    Initialize QueryableOperation instance
    // Retrieve data manager value
    DataManagerRequest DataManagerParams = DataManagerRequest.Value;
    // Handling paging operation.
    if (DataManagerParams.Skip != 0)
    {
        // Paging
        DataSource = queryableOperation.PerformSkip(DataSource,
            DataManagerParams.Skip);
    }
    if (DataManagerParams.Take != 0)
    {
        DataSource = queryableOperation.PerformTake(DataSource,
            DataManagerParams.Take);
    }
}
```



```

}
// Return data based on the request
return new { result = DataSource, count = totalRecordsCount };
}
// Model for handling data manager requests
public class DataManager
{
public required DataManagerRequest Value { get; set; }
}

```

### APP.COMPONENT.TS

```

import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
selector: 'app-root',
templateUrl: './app.component.html',
})
export class AppComponent {
public data?: DataManager;
@ViewChild('grid')
public grid?: GridComponent;
ngOnInit(): void {
this.data = new DataManager({
url: 'https://localhost:xxxx/api/Grid', // Replace your hosted link
adaptor: new WebMethodAdaptor()
});
}
}

```

### APP.COMPONENT.HTML

```

<ejs-grid #grid [dataSource]='data' allowPaging='true' height="320">
<e-columns>
<e-column field='OrderID' headerText='Order ID' isPrimaryKey=true
width='150'></e-column>
<e-column field='CustomerID' headerText='Customer Name' width='150'></e-
column>
<e-column field='ShipCity' headerText='ShipCity' width='150'
textAlign='Right'></e-column>
</e-columns>
</ejs-grid>

```

### APP.MODULE.TS

```

import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
declarations: [
AppComponent

```

```

],
imports: [
  BrowserModule, HttpClientModule,
  AppRoutingModule, GridModule
],
providers: [PageService],
bootstrap: [AppComponent]
})
export class AppModule { }

```

### Handling CRUD operations

The Syncfusion Angular Grid Component seamlessly integrates CRUD (Create, Read, Update, Delete) operations with server-side controller actions through specific properties: [insertUrl](#), [removeUrl](#), [updateUrl](#), [crudUrl](#), and [batchUrl](#). These properties enable the grid to communicate with the data service for every grid action, facilitating server-side operations.

### CRUD Operations Mapping

CRUD operations within the grid can be mapped to server-side controller actions using specific properties:

1. **insertUrl**: Specifies the URL for inserting new data.
2. **removeUrl**: Specifies the URL for removing existing data.
3. **updateUrl**: Specifies the URL for updating existing data.
4. **crudUrl**: Specifies a single URL for all CRUD operations.
5. **batchUrl**: Specifies the URL for batch editing.

To enable editing in Angular Grid component, refer to the editing [documentation](#). In the below example, the inline edit [mode](#) is enabled and [toolbar](#) property is configured to display toolbar items for editing purposes.

### APP.COMPONENT.TS

```

import { Component, ViewChild } from '@angular/core';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, EditSettingsModel, ToolbarItems } from
 '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
      insertUrl: 'https://localhost:xxxx/api/grid/Insert',
      updateUrl: 'https://localhost:xxxx/api/grid/Update',
      removeUrl: 'https://localhost:xxxx/api/grid/Remove',
      adaptor: new WebMethodAdaptor()
    });
  }
}

```

```
});
this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel'];
this.editSettings = { allowAdding: true, allowDeleting: true, allowEditing:
true, mode: 'Batch' };
}
}
```

#### APP.COMPONENT.HTML

```
<ejs-grid #grid [dataSource]='data' [toolbar]="toolbar"
[editSettings]="editSettings">
<e-columns>
<e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right' isPrimaryKey="true"></e-column>
<e-column field='CustomerID' headerText='Customer ID' width='160'></e-
column>
<e-column field='ShipCity' headerText='Ship City' width='150'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width='150'></e-
column>
</e-columns>
</ejs-grid>
```

#### APP.MODULE.TS

```
import { HttpClientModule } from '@angular/common/http';
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { EditService, GridModule, ToolbarService } from '@syncfusion/ej2-
angular-grids';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
@NgModule({
declarations: [
AppComponent
],
imports: [
BrowserModule, HttpClientModule,
AppRoutingModule, GridModule
],
providers: [EditService, ToolbarService],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Normal/Inline editing is the default edit [mode](#) for the Grid component. To enable CRUD operations, ensure that the [isPrimaryKey](#) property is set to **true** for a specific Grid column, ensuring that its value is unique.

The below class is used to structure data sent during CRUD operations.

```
`cs
public class CRUDModel<T> where T : class
{
```

```

public string? action { get; set; }
public string? keyColumn { get; set; }
public object? key { get; set; }
public T? value { get; set; }
public List<T>? added { get; set; }
public List<T>? changed { get; set; }
public List<T>? deleted { get; set; }
public IDictionary<string, object>? @params { get; set; }
}
`

```

### Insert operation:

To insert a new record, utilize the [insertUrl](#) property to specify the controller action mapping URL for the insert operation. The newly added record details are bound to the **newRecord** parameter.

```

▼ Request Payload      view source
  ▼ {value: {OrderID: 123, CustomerID: "ANATR", ShipCity: "Berlin", ShipCountry: "Denmark"},...}
    action: "insert"
    ▼ value: {OrderID: 123, CustomerID: "ANATR", ShipCity: "Berlin", ShipCountry: "Denmark"}
      CustomerID: "ANATR"
      OrderID: 123
      ShipCity: "Berlin"
      ShipCountry: "Denmark"

```

```
`cs
```

```
/// <summary>
```

```
/// Inserts a new data item into the data collection.
```

```
/// </summary>
```

```
/// <param name="newRecord">It contains the new record detail which is need to be
inserted.</param>
```

```
/// <returns>Returns void</returns>
```

```
[HttpPost]
```

```
[Route("api/[controller]/Insert")]
```

```
public void Insert([FromBody] CRUDModel<OrdersDetails> newRecord)
```

```
{
```

```
// Check if new record is not null
```

```
if (newRecord.value != null)
```

```
{
```

```
// Insert new record
```

```

OrdersDetails.GetAllRecords().Insert(0, newRecord.value);
}
}
,

```

### Update operation:

For updating existing records, utilize the [updateUrl](#) property to specify the controller action mapping URL for the update operation. The updated record details are bound to the **updatedRecord** parameter.

✕ Headers **Payload** Preview Response Initiator Timing

▼ Request Payload view source

```

{value: {orderId: 10004, customerID: "BLONP", employeeID: 4, freight: 5.3, shipCity: "Marseille",...},...}
  action: "update"
  key: 10004
  keyColumn: "OrderID"
  value: {orderId: 10004, customerID: "BLONP", employeeID: 4, freight: 5.3, shipCity: "Marseille",...}
    CustomerID: "ANATR"
    OrderID: 10004
    ShipCity: "Marseille"
    ShipCountry: "Austria"
    customerID: "BLONP"
    employeeID: 4
    freight: 5.3
    orderDate: "1930-10-21T18:30:00.000Z"
    orderID: 10004
    shipAddress: "Magazinweg 7"
    shipCity: "Marseille"
    shipCountry: "Austria"
    shipName: "Ernst Handel"
    shippedDate: "1996-12-29T18:30:00.000Z"
    verified: false

```

```

`cs
/// <summary>
/// Update a existing data item from the data collection.
/// </summary>
/// <param name="updatedRecord">It contains the updated record detail which is need to be
updated.</param>
/// <returns>Returns void</returns>
[HttpPost]
[Route("api/[controller]/Update")]
public void Update([FromBody] CRUDModel<OrdersDetails> updatedRecord)
{
// Retrieve updated order
var updatedOrder = updatedRecord.value;
if (updatedOrder != null)

```


```

{
// Find existing record
var data = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID == updatedOrder.OrderID);
if (data != null)
{
// Update existing record
data.OrderID = updatedOrder.OrderID;
data.CustomerID = updatedOrder.CustomerID;
data.ShipCity = updatedOrder.ShipCity;
data.ShipCountry = updatedOrder.ShipCountry;
// Update other properties similarly
}
}
}
`cs

```

### Delete operation

To delete existing records, use the [removeUrl](#) property to specify the controller action mapping URL for the delete operation. The primary key value of the deleted record is bound to the **deletedRecord** parameter.



```

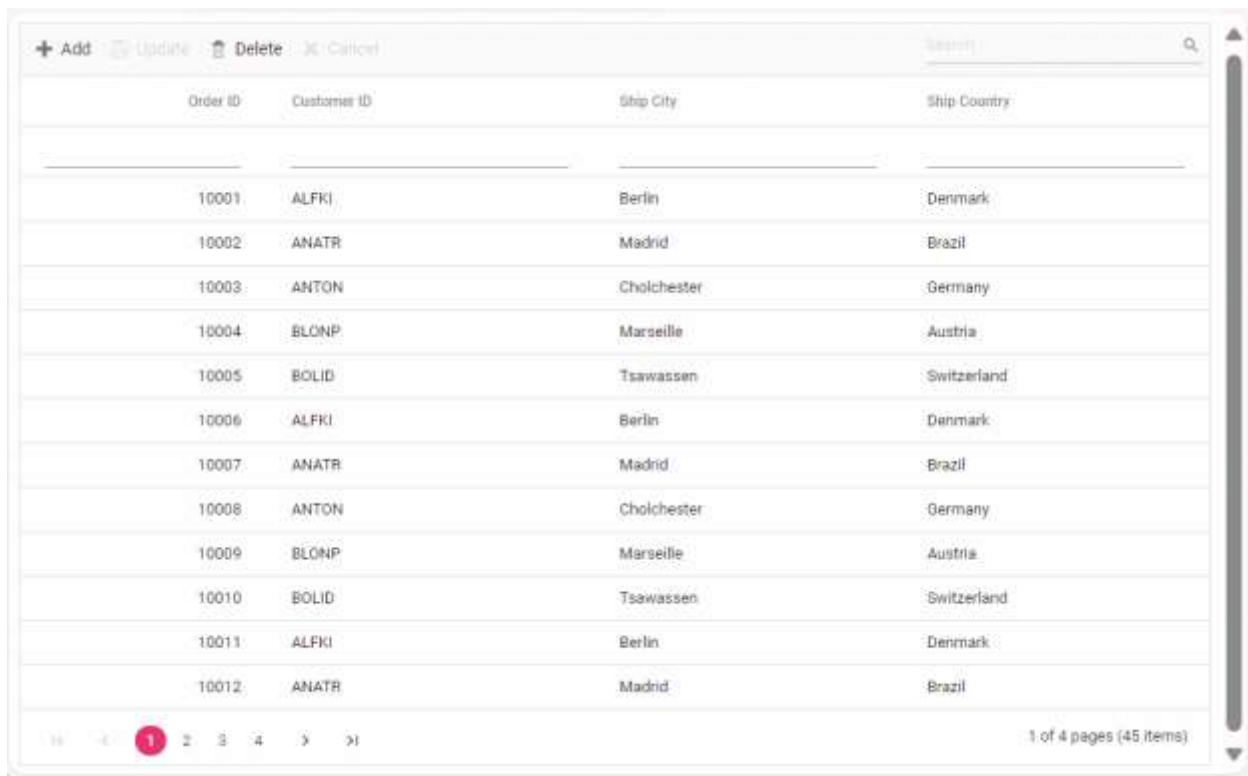
`cs
/// <summary>
/// Remove a specific data item from the data collection.
/// </summary>
/// <param name="deletedRecord">It contains the specific record detail which is need to be
removed.</param>
/// <return>Returns void</return>
[HttpPost]
[Route("api/[controller]/Remove")]

```

```

public void Remove([FromBody] CRUDModel<OrdersDetails> deletedRecord)
{
    int orderId = int.Parse(deletedRecord.key.ToString()); // get key value from the deletedRecord
    var data = OrdersDetails.GetAllRecords().FirstOrDefault(orderData => orderData.OrderID == orderId);
    if (data != null)
    {
        // Remove the record from the data collection
        OrdersDetails.GetAllRecords().Remove(data);
    }
}

```



Order ID	Customer ID	Ship City	Ship Country
10001	ALFKI	Berlin	Denmark
10002	ANATR	Madrid	Brazil
10003	ANTON	Cholchester	Germany
10004	BLONP	Marseille	Austria
10005	BOLID	Tsawassen	Switzerland
10006	ALFKI	Berlin	Denmark
10007	ANATR	Madrid	Brazil
10008	ANTON	Cholchester	Germany
10009	BLONP	Marseille	Austria
10010	BOLID	Tsawassen	Switzerland
10011	ALFKI	Berlin	Denmark
10012	ANATR	Madrid	Brazil

### Single method for performing all CRUD operations

Using the `crudUrl` property, the controller action mapping URL can be specified to perform all the CRUD operation at server-side using a single method instead of specifying separate controller action method for CRUD (insert, update and delete) operations.

The following code example describes the above behavior.

```

`ts
import { Component, ViewChild } from '@angular/core';

```

```

import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
import { GridComponent, EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent {
  public data?: DataManager;
  @ViewChild('grid')
  public grid?: GridComponent;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
      crudUrl: 'https://localhost:xxxx/api/grid/CrudUpdate',
      adaptor: new WebMethodAdaptor()
    });
    this.toolbar = ['Add', 'Edit', 'Update', 'Delete', 'Cancel'];
    this.editSettings = { allowAdding: true, allowDeleting: true, allowEditing: true, mode: 'Normal' };
  }
}
`cs
[HttpPost]
[Route("api/[controller]/CrudUpdate")]
public void CrudUpdate([FromBody] CRUDModel<OrdersDetails> request)
{
  // perform update operation
  if (request.action == "update")
  {
    var orderValue = request.value;

```



```

OrdersDetails existingRecord = OrdersDetails.GetAllRecords().Where(or => or.OrderID ==
orderValue.OrderID).FirstOrDefault();
existingRecord.OrderID = orderValue.OrderID;
existingRecord.CustomerID = orderValue.CustomerID;
existingRecord.ShipCity = orderValue.ShipCity;
}
// perform insert operation
else if (request.action == "insert")
{
OrdersDetails.GetAllRecords().Insert(0, request.value);
}
// perform remove operation
else if (request.action == "remove")
{
OrdersDetails.GetAllRecords().Remove(OrdersDetails.GetAllRecords().Where(or => or.OrderID ==
int.Parse(request.key.ToString()))).FirstOrDefault());
}
}
,

```

### Batch operation

To perform batch operation, define the edit [mode](#) as **Batch** and specify the `batchUrl` property in the DataManager. Use the **Add** toolbar button to insert new row in batch editing mode. To edit a cell, double-click the desired cell and update the value as required. To delete a record, simply select the record and press the **Delete** toolbar button. Now, all CRUD operations will be executed in single request. Clicking the **Update** toolbar button will update the newly added, edited, or deleted records from the OrdersDetails table using a single API POST request.

`ts

[app.component.ts]

```

import { Component, ViewChild } from '@angular/core';
import { GridComponent, ToolbarItems, EditSettingsModel } from '@syncfusion/ej2-angular-grids';
import { DataManager, WebMethodAdaptor } from '@syncfusion/ej2-data';
@Component({
selector: 'app-root',
template: `<ejs-grid #grid [dataSource]='data' [editSettings]='editSettings' [toolbar]='toolbar'>
<e-columns>

```

```

<e-column field='OrderID' headerText='Order ID' width='90' textAlign='Right' isPrimaryKey='true'></e-
column>
<e-column field="CustomerID" headerText="Customer Name" width="100"></e-column>
<e-column field='ShipCountry' headerText='Ship Country' width=100></e-column>
</e-columns>
</ejs-grid>`
})
export class AppComponent {
  @ViewChild('grid')
  public grid?: GridComponent;
  public data?: DataManager;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public orderIDRules?: object;
  public customerIDRules?: object;
  ngOnInit(): void {
    this.data = new DataManager({
      url: 'https://localhost:xxxx/api/grid', // Replace your hosted link
      batchUrl: 'https://localhost:xxxx/api/grid/BatchUpdate',
      adaptor: new WebMethodAdaptor()
    });
    this.editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Batch' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel', 'Search'];
    this.orderIDRules = { required: true };
    this.customerIDRules = { required: true, minLength: 3 };
  }
}
`cs
[HttpPost]
[Route("api/[controller]/BatchUpdate")]
public IActionResult BatchUpdate([FromBody] CRUDModel<OrdersDetails> batchOperation)
{

```

```
if (batchOperation.added != null)
{
    foreach (var addedOrder in batchOperation.added)
    {
        OrdersDetails.GetAllRecords().Insert(0, addedOrder);
    }
}
if (batchOperation.changed != null)
{
    foreach (var changedOrder in batchOperation.changed)
    {
        var existingOrder = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID ==
changedOrder.OrderID);
        if (existingOrder != null)
        {
            existingOrder.CustomerID = changedOrder.CustomerID;
            existingOrder.ShipCity = changedOrder.ShipCity;
            // Update other properties as needed
        }
    }
}
if (batchOperation.deleted != null)
{
    foreach (var deletedOrder in batchOperation.deleted)
    {
        var orderToDelete = OrdersDetails.GetAllRecords().FirstOrDefault(or => or.OrderID ==
deletedOrder.OrderID);
        if (orderToDelete != null)
        {
            OrdersDetails.GetAllRecords().Remove(orderToDelete);
        }
    }
}
return Json(batchOperation);
```

```

}
,

```

You can find the complete sample for the WebMethodAdaptor in [GitHub](#) link.

### Adaptive in Angular Grid component

The Grid user interface (UI) was redesigned to provide an optimal viewing experience and improve usability on small screens. This interface will render the filter, sort, column chooser, column menu(supports only when the `rowRenderingMode` as Horizontal) and edit dialogs adaptively and have an option to render the grid row elements in the vertical direction.

#### Render adaptive dialogs

The Syncfusion Angular Grid offers a valuable feature for rendering adaptive dialogs, specifically designed to enhance the user experience on smaller screens. This feature proves especially useful for optimizing the interface on devices with limited screen real estate. The functionality is achieved by enabling the [enableAdaptiveUI](#) property, allowing the grid to render filter, sort, and edit dialogs in full-screen mode.

The following sample demonstrates how to enable and utilize adaptive dialogs in the Syncfusion Angular Grid:

#### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'

```

```

import { PageService, SortService, FilterService, EditService,
ToolbarService, AggregateService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
imports: [

    GridModule
],
providers: [PageService,
            SortService,
            FilterService,
            EditService,
            ToolbarService,
            AggregateService],
standalone: true,
selector: 'app-root',
template: `<div class="e-adaptive-demo e-bigger">
    <div class="e-mobile-layout">
        <div class="e-mobile-content">
            <ejs-grid #adaptive id="adaptivebrowser"
[dataSource]='data' enableAdaptiveUI='true'
height='100%' allowPaging='true' allowFiltering='true'
allowSorting='true' [editSettings]='editSettings'
[filterSettings]='filterSettings' [toolbar]='toolbar'
(load)='onLoad()'>
                <e-columns>
                    <e-column field='SNO' headerText='S NO' width='150'
isPrimaryKey='true' [validationRules]='orderidrules'>
                        </e-column>
                    <e-column field='Model' headerText='Model'
width='200' editType='dropdownedit'
[validationRules]='customeridrules'>
                        </e-column>
                    <e-column field='Developer' headerText='Developer'
width='200' [validationRules]='customeridrules' [filter]='menuFilter'></e-
column>
                        <e-column field='ReleaseDate' headerText='Released
Date' width='200' type='date' format='yMMM' editType='datepickeredit'>
                            <e-column field='AndroidVersion' headerText='Android
Version' width='200' [validationRules]='customeridrules'
[filter]='checkboxFilter'></e-column>
                                </e-column>
                            </e-columns>
                        </ejs-grid>
                    </div>
                </div>
                <br />
                <div class="datalink">Source:
                    <a
href="https://en.wikipedia.org/wiki/List_of_Android_smartphones"
target="_blank">Wikipedia: List of Android
smartphones</a>
                </div>
            </div>`
})

```

```

export class AppComponent implements OnInit {
  @ViewChild('adaptive')
  public grid?: GridComponent;
  public data?: object[];
  public editSettings?: Object;
  public toolbar?: string[];
  public orderidrules?: Object;
  public customeridrules?: Object;
  public filterSettings?: Object;
  public menuFilter?: Object;
  public checkboxFilter?: Object;
  ngOnInit(): void {
    this.data = data;
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Dialog' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
'Search'];
    this.orderidrules = { required: true, number: true };
    this.customeridrules = { required: true };
    this.filterSettings = { type: 'Excel' };
    this.menuFilter = {
      type: 'Menu'
    };
    this.checkboxFilter = {
      type: 'CheckBox'
    };
  }
  public onLoad(): void {
    (this.grid as GridComponent).adaptiveDlgTarget =
document.getElementsByClassName('e-mobile-content')[0] as HTMLElement;
  }
}

```

### MAIN.TS

```

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

```

### Vertical row rendering

The Syncfusion Angular Grid introduces the feature of vertical row rendering, allowing you to display row elements in a vertical order. This is particularly useful for scenarios where a vertical presentation enhances data visibility. This is achieved by setting the [rowRenderingMode](#) property to the value **Vertical**.

The default row rendering mode is **Horizontal**.

The following sample demonstrates how to dynamically change the row rendering mode between **Vertical** and **Horizontal** based on a DropDownList selection:

### APP.COMPONENT.TS

```

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

```

```

import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, EditService,
ToolbarService, AggregateService } from '@syncfusion/ej2-angular-grids'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent, RowRenderingDirection } from '@syncfusion/ej2-
angular-grids';
import { data } from './datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [

    GridModule,
    DropDownListModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    EditService,
    ToolbarService,
    AggregateService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex; padding: 0px 0px 20px 200px">
      <label style="padding: 30px 17px 0 0;"> Select row rendering
mode :</label>
      <ejs-dropdownlist #dropdown style="padding: 26px 0 0 0"
index="0" width="150"
[dataSource]="dropDownData" (change)="changeAlignment($event)">
    </ejs-dropdownlist>
    </div>
    <div class="e-adaptive-demo e-bigger">
      <div class="e-mobile-layout">
        <div class="e-mobile-content">
          <ejs-grid #adaptive id="adaptivebrowser"
[dataSource]='data' enableAdaptiveUI='true' [rowRenderingMode]="rowMode"
height='100%' allowPaging='true' allowFiltering='true'
allowSorting='true' [editSettings]='editSettings'
[filterSettings]='filterSettings' [toolbar]='toolbar'
(load)='onLoad()'>
            <e-columns>
              <e-column field='SNO' headerText='S NO' width='150'
isPrimaryKey='true' [validationRules]='orderidrules'>
                </e-column>
              <e-column field='Model' headerText='Model'
width='200' editType='dropdownedit'
[validationRules]='customeridrules'>
                </e-column>
              <e-column field='Developer' headerText='Developer'
width='200' [validationRules]='customeridrules' [filter]='menuFilter'></e-
column>
              <e-column field='ReleaseDate' headerText='Released
Date' width='200' type='date' format='yMMM' editType='datepickeredit'>
              <e-column field='AndroidVersion' headerText='Android
Version' width='200' [validationRules]='customeridrules'
[filter]='checkboxFilter'></e-column>

```

```

        </e-column>
      </e-columns>
    <e-aggregates>
      <e-aggregate>
        <e-columns>
          <e-column type="Count" field="Model" >
            <ng-template #footerTemplate let-
data>Total Models: {{data.Count}}</ng-template>
          </e-column>
        </e-columns>
      </e-aggregate>
    </e-aggregates>
  </ejs-grid>
</div>
</div>
<br />
<div class="datalink">Source:
  <a
href="https://en.wikipedia.org/wiki/List_of_Android_smartphones"
target="_blank">Wikipedia: List of Android
smartphones</a>
</div>
</div>`
  ))
export class AppComponent implements OnInit {
  @ViewChild('adaptive')
  public grid?: GridComponent;
  public data?: object[];
  public editSettings?: Object;
  public toolbar?: string[];
  public orderidrules?: Object;
  public customeridrules?: Object;
  public filterSettings?: Object;
  public menuFilter?: Object;
  public checkboxFilter?: Object;
  public rowMode?: string;
  public dropDownData: Object[] = [
    { text: 'Vertical', value: 'Vertical' },
    { text: 'Horizontal', value: 'Horizontal' },
  ];
  ngOnInit(): void {
    this.data = data;
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Dialog' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
'Search'];
    this.orderidrules = { required: true, number: true };
    this.customeridrules = { required: true };
    this.filterSettings = { type: 'Excel' };
    this.menuFilter = {
      type: 'Menu'
    };
    this.checkboxFilter = {
      type: 'CheckBox'
    };
  }
  public changeAlignment(args: ChangeEventArgs): void {

```



```
(this.grid as GridComponent).rowRenderingMode = (args.value as RowRenderingDirection);
    }
    public onLoad(): void {
        (this.grid as GridComponent).adaptiveDlgTarget =
        document.getElementsByClassName('e-mobile-content')[0] as HTMLElement;
    }
}
```

### MAIN.TS

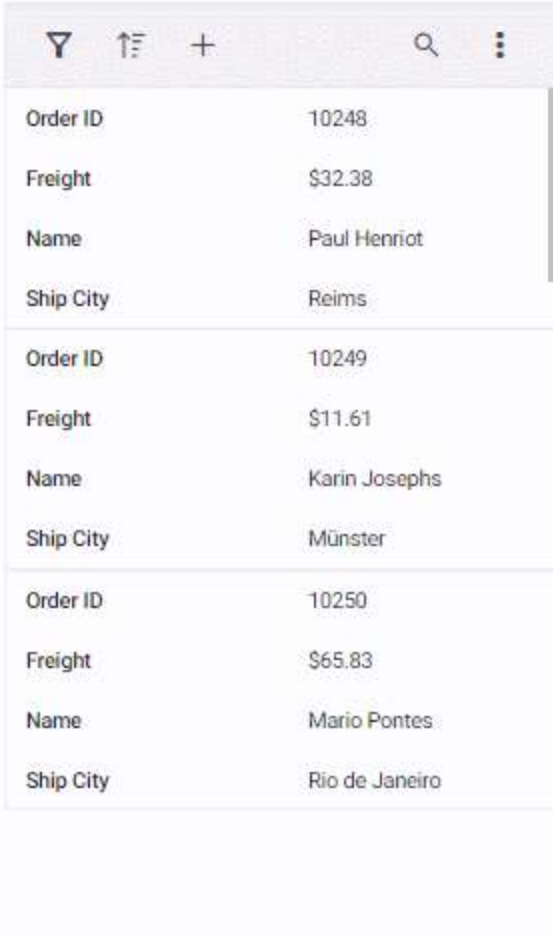
```
import { bootstrapApplication } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import 'zone.js';
bootstrapApplication(AppComponent).catch((err) => console.error(err));
```

[enableAdaptiveUI](#) property must be enabled for vertical row rendering.

#### *Supported features by vertical row rendering*

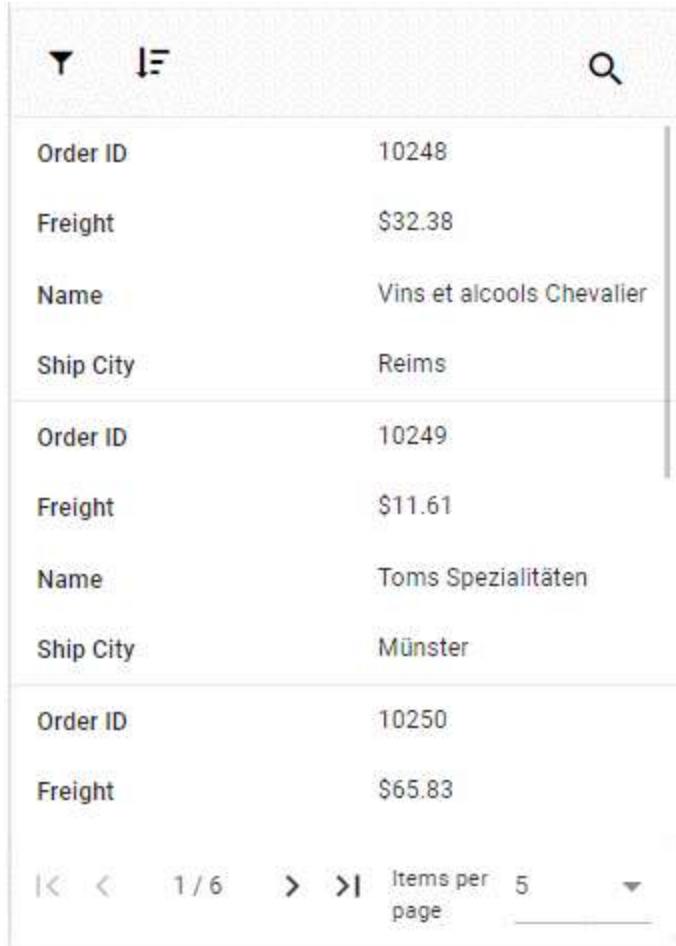
The following features are only supported in vertical row rendering:

- Paging, including Page size dropdown
- Sorting
- Filtering
- Selection
- Dialog Editing
- Aggregate
- Infinite scroll
- Toolbar - Options like **Add, Filter, Sort, Edit, Delete, Search**, and **Toolbar template** are available when their respective features are enabled. The toolbar dynamically includes a three-dotted icon, containing additional features like **ColumnChooser, Print, PdfExport, ExcelExport**, or **CsvExport**, once these features are enabled. Please refer to the following snapshot.



Order ID	10248
Freight	\$32.38
Name	Paul Henriot
Ship City	Reims
Order ID	10249
Freight	\$11.61
Name	Karin Josephs
Ship City	Münster
Order ID	10250
Freight	\$65.83
Name	Mario Pontes
Ship City	Rio de Janeiro

A snapshot of the adaptive grid displaying enabled paging along with a pager dropdown.



Order ID	10248
Freight	\$32.38
Name	Vins et alcools Chevalier
Ship City	Reims
Order ID	10249
Freight	\$11.61
Name	Toms Spezialitäten
Ship City	Münster
Order ID	10250
Freight	\$65.83

1/6 Items per page 5

The Column Menu feature, which includes grouping, sorting, autofit, filter, and column chooser, is exclusively supported for the Grid in **Horizontal** [rowRenderingMode](#).

See Also

- [Effective ways to utilize responsiveness](#)

## Performance tips for Angular DataGrid Component

This article is a comprehensive guide on improving the loading performance of the Angular DataGrid, especially when dealing with large datasets along with large number of columns. It provides valuable insights into the steps that need to be followed to bind a large data source without experiencing any performance degradations. By offering detailed explanations and actionable tips, this resource aims to empower readers with the knowledge and best practices necessary to optimize the performance of the Angular DataGrid during data binding, ensuring a smooth and efficient user experience.

### How to improve loading performance by binding large dataset

As you all know, a grid is made up of rows and columns. For instance, when you bind 10 rows and 10 columns, it means 100 elements will be rendered in the DOM (Document Object Model). So, it is recommended to render only a limited number of rows and columns to guarantee the best loading performance for the component.

### *Optimizing performance with paging*

To boost the performance efficiency of your application, especially when dealing with large datasets, it is advised to implement paging. [Paging](#) allows you to display grid data in segmented pages, facilitating easier navigation through substantial datasets. This feature proves particularly beneficial in enhancing the overall performance of your application. For more information on implementing paging, you can refer to the [documentation](#) section dedicated to this feature.

### *Optimizing performance with row virtualization or infinite scrolling*

To enhance your application's efficiency, especially when dealing with substantial datasets, it is recommended to either using [virtualization](#) or [infinite scrolling](#). Implementing these techniques can significantly reduce the load on your application and elevate its overall performance.

1. **Virtualization:** The Virtual scrolling feature in the Angular Data Grid enables the efficient handling and display of large volumes of data without compromising performance. This approach optimizes the rendering process by loading only the visible rows within the Grid viewport, rather than rendering the entire dataset simultaneously. For more information on implementing row virtualization , you can refer to the [documentation](#) section dedicated to this feature.
2. **Infinite scrolling:** The Infinite Scrolling feature in the Angular Data Grid is a powerful tool for seamlessly handling extensive data sets without compromising grid performance. It operates on a "load-on-demand" concept, ensuring that data is fetched only when needed. In the default infinite scrolling mode, a new block of data is loaded each time the scrollbar reaches the end of the vertical scroller. For more information on implementing infinite scrolling , you can refer to the [documentation](#) section dedicated to this feature.

### *Optimizing performance with column virtualization in large no of columns*

[Column virtualization](#) feature in the Angular Data Grid that allows you to optimize the rendering of columns by displaying only the columns that are currently within the viewport. It allows horizontal scrolling to view additional columns. This feature is particularly useful when dealing with grids that have a large number of columns, as it helps to improve the performance and reduce the initial loading time.

It is possible to enable both row and column virtualization. This feature allows for efficient handling of large datasets by dynamically loading only the visible rows and columns, optimizing performance and enhancing the overall responsiveness of the grid. For more information on implementing column virtualization , you can refer to the [documentation](#) section dedicated to this feature.

### *How to overcome browser height limitation in virtual scrolling*

#### [Documentation link](#)

How to improve loading performance by binding large data by showing custom text or element  
When integrating image or template elements into a column, it's recommended to utilize the [Column Template](#) feature rather than customizing the data through [rowDataBound](#) or [queryCellInfo](#) events. These events are triggered for each row and cell rendering, introducing delays in the component's rendering process. Moreover, rendering custom elements using these events may result in the persistence of rendered elements, potentially causing longer rendering times over time. By opting for the column template feature, you can efficiently meet this requirement without experiencing rendering delays and ensure a more streamlined rendering process.

### How to improve loading performance by referring individual script and CSS

To improve the performance of Syncfusion Grid component during the initial render as well as certain actions, suggested you to download the specific component scripts using CRG (Custom Resource Generator) to speed up the project. By default, the ej2.min.js script file contains all the Syncfusion component scripts. So, it will take some time to load the scripts to the project. Using [CRG](#), you can select the components which you want to use, and the modules for those components, then you can download the scripts and CSS for the selected components and use them as per your need.

#### [CRG website link](#)

So to improve the performance of grid during the initial rendering, suggested you to refer individual script and CSS.

### How to update cell values without frequent server calls

Efficiently update cell values without the need for frequent server calls, especially beneficial for live update scenarios. Even when the data is initially bound from the server, performing edit operations can be done without triggering a database refresh. Utilize the [setCellValue](#) method to update the DataGrid without affecting the database and only refresh the UI.

### How to optimize server-side data operations with adaptors

The Angular DataGrid provides support for various adaptors (OData, ODataV4, WebAPI, URL, etc.) to facilitate server-side data operations and CRUD functionalities. By leveraging these adaptors along with the **DataManager** component, you can seamlessly bind remote data sources to the grid and execute actions. During data operations like filtering, sorting, and paging, the corresponding action queries are generated as per the adaptor's requirements. It is crucial to handle these actions on the application end and return the processed data back to the grid. Refer to the documentation for comprehensive details. It's worth noting that for efficient data processing, the suggested order for returning processed data to the grid is as follows

- Filtering
- Sorting
- Aggregates
- Paging
- Grouping

### How to avoid MaxJsonLength error while passing large amount of records

The Angular Grid component is client-server based. So, we send the data as JSON object between client and server. The reported issue occurs due to the serialization of the large-sized JSON object. We need to increase the maximum length for serializing the large-sized JSON object. You have to alter the [MaxJsonLength](#) property on your web.config file or in the place of deserialization.

#### **Solution: 1**

```
`csharp
<configuration>
<system.web.extensions>
<scripting>
<webServices>
```

```
<jsonSerialization maxJsonLength="25000000"/>
</webServices>
</scripting>
</system.web.extensions>
</configuration>
`
```

### Solution : 2

```
`csharp
var serializer = new JavaScriptSerializer { MaxJsonLength = Int32.MaxValue };
`
```

### Optimizing Angular app performance with multiple grids and templates

The reported performance degradation issue is specifically linked to the Angular framework and is unrelated to the Syncfusion Grid.

When your application's DOM is populated with a large number of items, this problem arises as continuous change detection is applied (e.g., typing into an input continuously). For more information on common reasons for slowdowns in Angular apps, you can [refer](#) to the [documentation](#) links:

In [Angular](#), there are two default change detection strategies available:

- **Default:**

Utilizes the default **CheckAlways** strategy, where change detection is automatic until explicitly deactivated. For example, entering a value into a text box triggers continuous change detection for all template references, leading to the reported issue.

- **OnPush:**

Adopts the **CheckOnce** strategy, disabling automatic change detection until reactivated by setting the strategy to Default (CheckAlways). Enabling this strategy ensures that change detection triggers only for the input text box, rather than for all template references, overcoming the reported issue.

To address this, it's recommended to implement the OnPush change detection strategy in your application. This can be achieved by using the following code snippet:

```
`ts
@Component({
  selector: "app-root",
  templateUrl: "app.component.html",
  providers: [OrdersService],
  changeDetection: ChangeDetectionStrategy.OnPush
})
```

Using the OnPush strategy may lead to child components not being updated when the input changes. You can address this by referring to the following links,

- [OnPush Change Detection](#)
- [ApplicationRef - Tick](#)
- [Tick - Description](#)
- [DetectChange - Anchor](#)

### Microsoft excel limitation while exporting millions of records to excel file format

By default, Microsoft Excel supports only 1,048,576 records in an excel sheet. Hence it is not possible to export millions of records to excel. You can refer the [documentation](#) link for more details on Microsoft excel specifications and limits. So suggest to export the data in CSV (Comma-Separated Values) or other formats that can handle large datasets more efficiently than Excel.

## Columns in Angular Grid Component

In Syncfusion Angular Grid, Columns are fundamental elements that play a pivotal role in organizing and displaying data within your application. They serve as the building blocks for data presentation, allowing you to specify what data fields to show, how to format and style them, and how to enable various interactions within the grid.

### Column types

The Syncfusion Grid component allows you to specify the type of data that a column binds using the [columns.type](#) property. The [type](#) property is used to determine the appropriate [format](#), such as [number](#) or [date](#), for displaying the column data.

Grid supports the following column types:

- **string**: Represents a column that binds to string data. This is the default type if the [type](#) property is not defined.
- **number**: Represents a column that binds to numeric data. It supports formatting options for displaying numbers.
- **boolean**: Represents a column that binds to boolean data. It displays checkboxes for boolean values.
- **date**: Represents a column that binds to date data. It supports formatting options for displaying dates.
- **datetime**: Represents a column that binds to date and time data. It supports formatting options for displaying date and time values.
- **checkbox**: Represents a column that displays checkboxes.

Here is an example of how to specify column types in a grid using the types mentioned above.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
```

```

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

    GridModule
  ],
  providers: [PageService,
              SortService,
              FilterService,
              GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'>
    <e-columns>
      <e-column type='checkbox' width=90></e-column>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90 type='number'></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120 type='string'></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90 type='number'></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120 type='date'></e-column>
      <e-column field='ShippedDate' headerText='Shipped Date'
textAlign='Right' format='dd/MM/yyyy hh:mm' width=200 type='dateTime'></e-
column>
      <e-column field='Verified' headerText='Verified'
width='100' type='boolean' [displayAsCheckBox]='true'></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}

```

## 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 [type](#) is not defined, then it will be determined from the first record of the [dataSource](#).

\* In case if the first record of the [dataSource](#) is null/blank value for a column then it is necessary to define the [type](#) for that column. This is because the grid uses the column type to determine which filter dialog to display for that column

### *Difference between boolean type and checkbox type column*

1. Use the **boolean** column type when you want to bind boolean values from your datasource and/or edit boolean property values from your type.



2. Use the **checkbox** column type when you want to enable selection/deselection of the whole row.
3. When the grid column type is a **checkbox**, the selection type of the grid `selectionSettings` will be multiple. This is the default behavior.
4. If you have more than one column with the column type as a **checkbox**, the grid will automatically enable the other column's checkbox when selecting one column checkbox.

To learn more about how to render boolean values as checkboxes in a Syncfusion GridColumn, please refer to the [Render Boolean Values as Checkbox](#) section.

### Column Width

To adjust the column width in a Grid, you can use the `width` property within the `columns` property of the Grid configuration. This property enables you to define the column width in pixels or as a percentage. You can set the width to a specific value, like **100** for 100 pixels, or as a percentage value, such as **25%** for 25% of the available width.

1. Grid column width is calculated based on the sum of column widths. For example, a grid container with 4 columns and a width of 800 pixels will have columns with a default width of 200 pixels each.
2. If you specify widths for some columns but not others, the Grid will distribute the available width equally among the columns without explicit widths. For example, if you have 3 columns with widths of 100px, 200px, and no width specified for the third column, the third column will occupy the remaining width after accounting for the first two columns.
3. Columns with percentage widths are responsive and adjust their width based on the size of the grid container. For example, a column with a width of 50% will occupy 50% of the grid width and will adjust proportionally when the grid container is resized to a smaller size.
4. When you manually resize columns in Syncfusion Grid, a minimum width is set to ensure that the content within the cells remains readable. By default, the minimum width is set to 10 pixels if not explicitly specified for a column.
5. If the total width of all columns exceeds the width of the grid container, a horizontal scrollbar will automatically appear to allow horizontal scrolling within the grid.
6. When the columns are hidden using the column chooser, the width of the hidden columns is removed from the total grid width, and the remaining columns will be resized to fill the available space.
7. If the parent element has a fixed width, the grid component will inherit that width and occupy the available space. However, if the parent element does not have a fixed width, the grid component will adjust its width dynamically based on the available space.

### Supported types for column width

Syncfusion Grid supports the following three types of column width:

#### 1. Auto

The column width is automatically calculated based on the content within the column cells. If the content exceeds the width of the column, it will be truncated with an ellipsis (...) at the end. You can set the width for columns as **auto** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='auto'></e-column>
```

## 2. Percentage

The column width is specified as a percentage value relative to the width of the grid container. For example, a column width of 25% will occupy 25% of the total grid width. You can set the width for columns as **percentage** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='25%'></e-column>
```

## 3. Pixel

The column width is specified as an absolute pixel value. For example, a column width of 100px will have a fixed width of 100 pixels regardless of the grid container size. You can set the width for columns as **pixel** in your Grid configuration as shown below:

```
`html
```

```
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width='100'></e-column>
```

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' width='auto'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width='auto'></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=100></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width='30%'></e-column>
    </e-columns>
  </ejs-grid>`
})
```

```
export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
        this.data = data;
    }
}
```

### MAIN.TS

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

### Column formatting

Column formatting is a powerful feature in Syncfusion Grid that allows you to customize the display of data in grid columns. You can apply different formatting options to columns based on your requirements, such as displaying numbers with specific formats, formatting dates according to a specific locale, and using templates to format column values.

You can use the [columns.format](#) property to specify the format for column values.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
    imports: [
        GridModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' height="315px">
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
            <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
        </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
        this.data = data;
    }
}
```

**MAIN.TS**

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

- \* The grid uses the [Internalization](#) library to format values based on the specified format and culture.
- \* By default, the [number](#) and [date](#) values are formatted in **en-US** locale. You can localize the currency and date in different locale as explained [here](#).
- \* The available format codes may vary depending on the data type of the column.
- \* You can also customize the formatting further by providing a custom function to the **format** property, instead of a format string.
- \* Make sure that the format string is valid and compatible with the data type of the column, to avoid unexpected results.

*Number formatting*

Number formatting allows you to customize the display of numeric values in grid columns. You can use standard numeric format strings or custom numeric format strings to specify the desired format. The [columns.format](#) property can be used to specify the number format for numeric columns. For example, you can use the following format strings to format numbers:

Format | Description | Remarks

{ type:'date', format:'dd/MM/yyyy' } | 04/07/1996

{ type:'date', format:'dd.MM.yyyy' } | 04.07.1996

{ type:'date', skeleton:'short' } | 7/4/96

{ type: 'dateTime', format: 'dd/MM/yyyy hh:mm a' } | 04/07/1996 12:00 AM

{ type: 'dateTime', format: 'MM/dd/yyyy hh:mm:ss a' } | 07/04/1996 12:00:00 AM

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height="315px">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
```

```

        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' [format]='formatOptions'
headerText='Order Date' textAlign='Right' width=120></e-column>
        <e-column field='OrderDate' [format]='shipFormat'
headerText='Ship Date' textAlign='Right' width=150></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public formatOptions?: object;
        public shipFormat?: object;
        ngOnInit(): void {
            this.data = data;
            this.formatOptions = {type: 'date', format: 'dd/MM/yyyy'};
            this.shipFormat = { type: 'dateTime', format: 'dd/MM/yyyy hh:mm a'
        };
    }
}

```

### MAIN.TS

```

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

```

To learn more about date formatting, you can refer to [Date formatting](#).

#### *Format the date column based on localization*

You can also format the date column based on the localization settings of the user's browser. You can use the [format](#) property of the Grid columns along with the [locale](#) property to specify the desired date format based on the locale.

In this example, the format property specifies the date format as "yyyy-MMM-dd", and the locale property specifies the locale as "es-AR" for Spanish (Argentina).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, PageService } from '@syncfusion/ej2-angular-grids'
import { L10n, loadCldr, setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import cagregorian from './ca-gregorian.json';
import currencies from './currencies.json';
import numbers from './numbers.json';
import timeZoneNames from './timeZoneNames.json';
@Component({
  imports: [

    GridModule

```

```

    ],
    providers: [GroupService, PageService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [locale]='locale'
height='315px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
            <e-column field='Freight' headerText='Freight'
format='C2' textAlign='Right' width=150></e-column>
            <e-column field='OrderDate' headerText='OrderDate'
[format]='formatOptions' textAlign='Right' width=150></e-column>
            <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public formatOptions?: object;
    public locale: string = 'es-AR';
    ngOnInit(): void {
      setCulture('es-AR');
      setCurrencyCode('ARS');
      loadCldr(
        cagregorian,
        currencies,
        numbers,
        timeZoneNames
      );

      this.formatOptions= { type: 'date', format: 'yyyy-MMM-dd'};
      this.data = data;
    }
  }
}

```

## MAIN.TS

```

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

```

### Format template column value

Template columns in Grid provide a way to customize the appearance of column values using HTML templates. In addition to HTML markup, you can also use number formatting to format the value displayed in a template column. To format values in a column template, you can use Angular pipes and the [format](#) property. In this example, we are using the date pipe to format the **OrderDate** value as a date in the format **'dd/MMM/yyyy'**.

`ts

```

<e-column field='OrderDate' headerText='Order Date' textAlign='Right' width=120>
<ng-template #template let-data>
{{ data.OrderDate | date:'dd/MMM/yyyy' }}
</ng-template>
</e-column>
,

```

### APP.COMPONENT.TS

```

{% raw %}
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height="315px">
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
<e-column field='Freight' headerText='Freight' textAlign='Right' format='C2'
width=90></e-column>
<e-column field='OrderDate' headerText='Order Date' textAlign='Right'
width=120>
<ng-template #template let-data>
{{ data.OrderDate | date:'dd/MMM/yyyy' }}
</ng-template>
</e-column>
<e-column field='ShipCountry' headerText='Ship Country' textAlign='Right'
width=150></e-column>
</e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public formatOptions?: object;
  public shipFormat?: object;
  ngOnInit(): void {
    this.data = data;
  }
}
{% endraw %}

```

### 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 other Angular pipes, such as **currency**, **decimal**, **percent**, etc., to format other types of values in the column template based on your requirements.

### Custom formatting

Syncfusion Grid allows you to customize the formatting of data in the grid columns. You can apply custom formats to numeric or date columns to display data in a specific way according to the requirements. To apply custom formatting to grid columns in Syncfusion Grid, you can use the [format](#) property. Here's an example of how you can use custom formatting for numeric and date columns:

In the below example, the **numberFormatOptions** object is used as the **format** property for the **'Freight'** column to apply a custom numeric format with four decimal places. Similarly, the **dateFormatOptions** object is used as the **format** property for the **'OrderDate'** column to apply a custom date format displaying the date in the format of day-of-the-week, month abbreviation, day, and 2-digit year (e.g. Sun, May 8, '23).

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height="315px">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' [format]='numberFormatOptions' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' [format]='dateFormatOptions' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public numberFormatOptions?: object;
  public dateFormatOptions?: object;
  ngOnInit(): void {
    this.data = data;
    this.numberFormatOptions = {
      // Custom format for numeric columns
      format: '##.0000',
    };
    this.dateFormatOptions = {
      // Custom format for date columns
      type: 'Date',
      format: "EEE, MMM d, 'yy",
    };
  }
}
```



**MAIN.TS**

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

To learn more about custom formatting, you can refer to [Custom Date formatting](#) and [Custom Number formatting](#).

**Align the text of content**

You can align the text in the content of a Grid column using the [textAlign](#) property. This property allows you to specify the alignment of the text within the cells of a particular column in the Grid. By default, the text is aligned to the left, but you can change the alignment by setting the value of the [textAlign](#) property to one of the following options:

*Left: Aligns the text to the left (default). Center: Aligns the text to the center. Right: Aligns the text to the right. Justify: Align the text to the justify.*

Here is an example of using the [textAlign](#) property to align the text of a Grid column:

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent, Column } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
    DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 30px 17px 0 0;">Align the text for columns
    </label>
    <ejs-dropdownlist style="padding: 26px 0 0 0" index="0" width="100"
    [dataSource]="alignmentData" (change) ="changeAlignment($event)"></ejs-
    dropdownlist>
    </div>
    <ejs-grid #grid [dataSource]='data' height='315px' style="padding-
    top:20px">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID' type='number'
        width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        type='string' width=90></e-column>
```

```

        <e-column field='OrderDate' headerText='Order Date' type='date'
textAlign='Center' format='yMd' width=140></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
type='string' width=120></e-column>
    </e-columns>
</ejs-grid>,
}))
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid') public grid?: GridComponent;
    public alignmentData: Object[] = [
        { text: 'Left', value: 'Left' },
        { text: 'Right', value: 'Right' },
        { text: 'Center', value: 'Center' },
        { text: 'Justify', value: 'Justify' },
    ];
    public changeAlignment(args: ChangeEventArgs): void {
        (this.grid as GridComponent).columns.forEach((col: Column) => {
            col.textAlign = args.value as string;
        });
        (this.grid as GridComponent).refreshColumns();
    }
    ngOnInit(): void {
        this.data = data;
    }
}

```

## MAIN.TS

```

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

```

\* The `textAlign` property only changes the alignment content not the column header. If you want to align both the column header and content, you can use the [headerTextAlign](#) property.

## Render boolean value as checkbox

The Grid component allows you to render boolean values as checkboxes in columns. This can be achieved by using the [displayAsCheckBox](#) property, which is available in the [columns](#). This property is useful when you have a boolean column in your Grid and you want to display the values as checkboxes instead of the default text representation of **true** or **false**.

To enable the rendering of boolean values as checkboxes, you need to set the `displayAsCheckBox` property of the `columns` to **true**.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
    imports: [

```

```

        GridModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [height]='315'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign= 'Right' width=100 format= 'C2'></e-column>
            <e-column field='Verified' headerText='Verified'
[displayAsCheckBox]="true" width=150></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
      this.data = data;
    }
  }
}

```

### MAIN.TS

```

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

```

\* The `displayAsCheckBox` property is only applicable to boolean values in Grid columns.

\* When `displayAsCheckBox` is set to **true**, the boolean values will be rendered as checkboxes in the Grid column, with checked state indicating **true** and unchecked state indicating **false**.

### How to prevent checkbox in the blank row

To prevent the checkbox in the blank row of the Grid, even if the `displayAsCheckBox` property is set to true for that column, you can use the `rowDataBound` event and check for empty or null values in the row data. If all the values in the row are empty or null, you can set the inner HTML of the corresponding cell to an empty string to hide the checkbox.

Here is an example of how you can prevent a checkbox from being displayed in a blank row in a Grid:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { RowDataBoundEventArgs, GridComponent, Column } from
 '@syncfusion/ej2-angular-grids';
@Component({

```

```

imports: [
    GridModule
],
providers: [PageService],
standalone: true,
selector: 'app-root',
template: `<ejs-grid #grid [dataSource]='data' [height]='315'
    (rowDataBound)='rowDataBound($event)'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=80></e-column>
            <e-column field='Verified' headerText='Verified'
width=130 displayAsCheckBox="true"></e-column>
        </e-columns>
    </ejs-grid>`
))
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    rowDataBound(args: RowDataBoundEventArgs) {
        let count = 0;
        let data: { [key: string]: object | string } = args.data as { [key:
string]: object | string };
        let keys = Object.keys(data);
        for (let i = 0; i < keys.length; i++) {
            if (data[keys[i]] == null || data[keys[i]] == '' ||
data[keys[i]] == undefined) {
                count++;
            }
        }
        if (count == keys.length) {
            for (let i = 0; i < (this.grid as GridComponent).columns.length;
i++) {
                if (((this.grid as GridComponent).columns[i] as
Column).displayAsCheckBox) {
                    (args.row as Element).children[i].innerHTML = '';
                }
            }
        }
    }
}
}

```

### MAIN.TS

```

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

```

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

### AutoFit columns

The Grid has a feature that allows to automatically adjust column widths based on the maximum content width of each column when you double-click on the resizer symbol located in a specific column header. This feature ensures that all data in the grid rows is displayed without wrapping. To use this feature, you need to inject the **ResizeService** in the provider section of **AppModule** and enable the resizer symbol in the column header by setting the [allowResizing](#) property to true in the grid.

#### *Resizing a column to fit its content using autoFit method*

The [autoFitColumns](#) method resizes the column to fit the widest cell's content without wrapping. You can autofit specific columns at initial rendering by invoking the `autoFitColumns` method in [dataBound](#) event.

To use `autoFitColumns` method, you need to inject **ResizeService** in the provider section of **AppModule**.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ResizeService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ResizeService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' height='315px'
gridLines='Both' (dataBound)='dataBound()' >
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=120></e-column>
<e-column field='ShipName' headerText='Ship Name' width=80></e-column>
<e-column field='ShipAddress' headerText='Ship Address' width=120></e-
column>
<e-column field='ShipCity' headerText='Ship City' width=100></e-column>
</e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  dataBound() {
```

```
(this.grid as GridComponent).autoFitColumns(['ShipAddress',
'ShipName']);
}
```

### 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 autofit all the columns by invoking the `autoFitColumns` method without specifying column names.

#### *AutoFit columns with empty space*

The Autofit feature is utilized to display columns in a grid based on the defined width specified in the columns declaration. If the total width of the columns is less than the width of the grid, this means that white space will be displayed in the grid instead of the columns auto-adjusting to fill the entire grid width.

You can enable this feature by setting the `autoFit` property set to `true`. This feature ensures that the column width is rendered only as defined in the Grid column definition.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ResizeService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ResizeService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' allowResizing= 'true' height=
'400' width= '850' autoFit= 'true'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' minWidth='100' width='150'
maxWidth='200' textAlign='Right'></e-column>
<e-column field='CustomerID' headerText='Customer ID' minWidth='8'
width='150'></e-column>
<e-column field='Freight' headerText='Freight' minWidth='8' width='120'
format='C2' textAlign='Right'></e-column>
<e-column field='ShipCity' headerText='Ship City' [allowResizing] = 'false'
width='150' textAlign='Right'></e-column>
<e-column field='ShipCountry' headerText='Ship Country' minWidth='8'
width='150'></e-column>
</e-columns>
</ejs-grid>`
```

```

    })
    export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
        public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    }

```

## MAIN.TS

```

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

```

If any one of the column width is undefined, then the particular column will automatically adjust to fill the entire width of the grid table, even if you have enabled the `autoFit` property of grid.

### *AutoFit columns when changing column visibility using column chooser*

In Syncfusion Grid, you can auto-fit columns when the column visibility is changed using the column chooser. This can be achieved by calling the `autoFitColumns` method in the `actionComplete` event. By using the `requestType` property in the event arguments, you can differentiate between different actions, and then call the `autoFitColumns` method when the request type is `columnState`.

Here's an example code snippet in Angular that demonstrates how to auto fit columns when changing column visibility using column chooser:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ResizeService, ToolbarService, ColumnChooserService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ActionEventArgs, GridComponent, ToolbarItems } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ResizeService, ToolbarService, ColumnChooserService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]="data" height="350px"
[allowPaging]="true"
[toolbar]="toolbarItems" (actionComplete)="onActionComplete($event)"
[showColumnChooser]= 'true'>
    <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>

```

```

    <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
    <e-column field='ShipName' headerText='Ship Name' width=80></e-column>
    <e-column field='ShipAddress' headerText='Ship Address' width=120></e-
column>
    <e-column field='ShipCity' headerText='Ship City' width=100></e-column>
  </e-columns>
</ejs-grid>
`
,
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public toolbarItems?: ToolbarItems[];
  ngOnInit(): void {
    this.data = data;
    this.toolbarItems = ['ColumnChooser'];
  }
  public onActionComplete({ requestType }: ActionEventArgs): void {
    if (requestType === 'columnstate') {
      (this.grid as GridComponent).autoFitColumns();
    }
  }
}

```

## MAIN.TS

```

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

```

### *AutoFit columns with specific rows*

To adjust the column widths of a specific range of rows based on their content, you can use the [autoFitColumns](#) method by simply passing the second and third parameters (optional) as the start and end index for the column you want to fit. You can autofit specific columns at initial rendering by invoking the [autoFitColumns](#) method in [dataBound](#) event.

This feature will calculate the appropriate width based on the maximum content width of the specified range of rows or the header text width. Subsequently, the maximum width of the content of the specified rows or header text will be applied to the entire column of the grid.

Here is an example of how to autofit columns with specific rows. The first parameter is an array containing the specific column field names, such as **Inventor**, **Number of INPADOC patents** and **Main fields of invention** is passed to apply the autofit functionality to these columns. After, the second parameter are start index is set to **1** and third parameter are end index is set to **3**. When specifying these start and end index, the autofit operation is applied only to the range of rows from 1 to 3 for column width adjustment.

## APP.COMPONENT.TS

```

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

```



```

import { GridModule, ResizeService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { inventoryData } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ResizeService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' height='315px'
[allowResizing]='true' (dataBound)='dataBound()' ` >
    <e-columns>
      <e-column field='Inventor' headerText='Inventor' textAlign='Right'
width=100 clipMode='EllipsisWithTooltip'></e-column>
      <e-column field='NumberofPatentFamilies' headerText='Number of Patent
Families' width=120></e-column>
      <e-column field='Country' headerText='Country' width=80></e-column>
      <e-column field='Number of INPADOC patents' headerText='Number of INPADOC
patents' width=150></e-column>
      <e-column field='Active' headerText='Active' width=100></e-column>
      <e-column field='Mainfieldsofinvention' headerText='Main fields of
invention' width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = inventoryData;
  }
  dataBound() {
    (this.grid as GridComponent).autoFitColumns(['Inventor', 'Number of
INPADOC patents', 'Mainfieldsofinvention'], 1, 3);
  }
}

```

## MAIN.TS

```

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

```

## Locked columns

The Syncfusion Grid allows you to lock columns, which prevents them from being reordered and moves them to the first position. This functionality can be achieved by setting the [column.lockColumn](#) property to **true**, which locks the column and moves it to the first position in the grid.

Here's an example of how you can use the `lockColumn` property to lock a column in the Syncfusion Grid:

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ReorderService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [ReorderService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowReordering]='true'
[allowSelection]='false' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='ShipCity' width=100 [lockColumn]='true'
[customAttributes]='customAttributes'></e-column>
      <e-column field='ShipName' width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public customAttributes?: object;
  ngOnInit(): void {
    this.data = data;
    this.customAttributes = {class: 'customcss'};
  }
}
```

**MAIN.TS**

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

**INDEX.HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Syncfusion Angular Grid</title>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <meta name="description" content="Typescript UI Controls" />

  <meta name="author" content="Syncfusion" />
```

```

<style>
#loader {
  color: #008cff;
  font-family: 'Helvetica Neue','calibiri';
  font-size: 16px;
  height: 40px;
  left: 45%;
  position: absolute;
  top: 45%;
  width: 30%;
}
#reorderMultipleCols {
  text-transform: none;
}
#reorderSingleCol {
  text-transform: none;
}
.e-grid .e-rowcell.customcss{
  background-color: #ecedee;
}
.e-grid .e-headercell.customcss{
  background-color: #ecedee;
}
</style>
</head>
<body style="margin-top: 125px">
  <app-root>
    <div id='loader'>Loading....</div>
  </app-root>
</body>
</html>

```

### Show or hide columns

The Syncfusion Grid control allows you to show or hide columns dynamically by using property or methods available in the grid. This feature can be useful when you want to customize the visibility of columns in the Grid based on the requirements.

#### Using property

You can show or hide columns in the Angular Grid using the [visible](#) property of each column. By setting the **visible** property to **true** or **false**, you can control whether the column should be visible or hidden in the grid. Here's an example of how to show or hide a column in the Angular Grid using the visible property:

In the below example, the **ShipCity** column is defined with **visible** property set to **false**, which will hide the column in the rendered grid.

#### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

```

```

        GridModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' height='315px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=140></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C' width=120></e-column>
            <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=140></e-column>
            <e-column field='ShipCity' headerText='Ship City'
[visible]='false' width=100></e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        ngOnInit(): void {
            this.data = data;
        }
    }
}

```

## MAIN.TS

```

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

```

\* Hiding a column using the `visible` property only affects the UI representation of the grid. The data for the hidden column will still be available in the underlying data source, and can be accessed or modified programmatically.

\* When a column is hidden, its width is not included in the calculation of the total grid width.

\* To hide a column permanently, you can set its `visible` property to `false` in the column definition, or remove the column definition altogether.

### Using methods

You can also show or hide columns in the Angular Grid using the [showColumns](#) and [hideColumns](#) methods of the grid component. These methods allow you to show or hide columns based on either the `headerText` or the `field` of the column.

### Based on header text

You can dynamically show or hide columns in the Grid based on the header text by invoking the `showColumns` or `hideColumns` methods. These methods take an array of column header texts as the first parameter, and the value `headerText` as the second parameter to specify that you are showing or hiding columns based on the header text.

Here's an example of how to show or hide a column based on the HeaderText in the Angular Grid:

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: ` <button id='show' ejs-button cssClass="e-info"
(click)='show()' > Show </button>
      <button id='hide' ejs-button cssClass="e-info"
(click)='hide()' > Hide </button>
      <ejs-grid #grid [dataSource]='data' [height]='280'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
          <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
          <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
          <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
        </e-columns>
      </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  show() {
    (this.grid as any).showColumns('Customer ID', 'headerText'); // show
    by HeaderText
  }
  hide() {
    (this.grid as any).hideColumns('Customer ID', 'headerText'); // hide
    by HeaderText
  }
}
```

#### **MAIN.TS**

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

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

### Based on field

You can dynamically show or hide columns in the Grid using external buttons based on the field by invoking the `showColumns` or `hideColumns` methods. These methods take an array of column fields as the first parameter, and the value `field` as the second parameter to specify that you are showing or hiding columns based on the field.

Here's an example of how to show or hide a column based on the field in the Angular Grid:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: ` <button id='show' ej-button cssClass="e-info"
(click)='show()'> Show </button>
                <button id='hide' ej-button cssClass="e-info"
(click)='hide()'> Hide </button>
                <ejs-grid #grid [dataSource]='data' [height]='280'>
                  <e-columns>
                    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
                    <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
                    <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
                    <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
                  </e-columns>
                </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  show() {
    (this.grid as GridComponent).showColumns('CustomerID', 'field'); //
show by field
  }
  hide() {
```

```
(this.grid as GridComponent).hideColumns('CustomerID', 'field'); //
hide by field
    }
}
```

### MAIN.TS

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

### Controlling Grid actions

You can control various actions such as filtering, grouping, sorting, resizing, reordering, editing, and searching for specific columns in the Syncfusion Angular Grid using the following properties:

- [allowEditing](#): Enables or disables editing for a column.
- [allowFiltering](#): Enables or disables filtering for a column.
- [allowGrouping](#): Enables or disables grouping for a column.
- [allowSorting](#): Enables or disables sorting for a column.
- [allowReordering](#): Enables or disables reordering for a column.
- [allowResizing](#): Enables or disables resizing for a column.
- [allowSearching](#): Enables or disables searching for a column.

Here is an example code that demonstrates how to control grid actions for specific columns:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, SortService, FilterService, GroupService,
ReorderService, ToolbarService, ResizeService, } from '@syncfusion/ej2-
angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService, PageService, SortService, ReorderService,
FilterService, ToolbarService, ResizeService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'
[allowPaging]='true' [allowSorting]='true' [allowFiltering]='true'
[allowReordering]='true' [allowResizing]='true' [toolbar]="toolbar"
[allowGrouping]='true'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width='100' [allowGrouping]="false" [allowResizing]=
'false'></e-column>
```

```

        <e-column field='CustomerID' headerText='Customer ID'
        textAlign='Left' width='150' [allowSorting]="false"></e-column>
        <e-column field='ShipCity' headerText='Ship City'
        textAlign='Left' width='150' [allowReordering]="false"></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
        textAlign='Left' width='150' [allowSearching]="false"></e-column>
        <e-column field='Freight' headerText='Freight'
        textAlign='Right' width='150' format='C2' [allowFiltering]="false"></e-
        column>
    </e-columns>
</ejs-grid>
`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbar: any;
        ngOnInit(): void {
            this.data = data;
            this.toolbar = ['Search'];
        }
    }
}

```

## MAIN.TS

```

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

```

## Customize column styles

Customizing the grid column styles allows you to modify the appearance of columns in the Grid control to meet your design requirements. You can customize the font, background color, and other styles of the columns. To customize the columns styles in the grid, you can use grid event, css, property or method support.

For more information check on this [documentation](#).

## Manipulating columns

The Syncfusion Grid for Angular provides powerful features for manipulating columns in a grid. This section explains how to access columns, update column definitions, and add/remove columns using Syncfusion Grid properties, methods, and events.

### Accessing Columns

To access columns in the Syncfusion Grid, you can use the following methods in the grid.

- [getColumns](#):

This method returns the array of columns defined in the grid.

```
`ts
```

```
let columns = this.grid.getColumns();
```

```
,
```



- [getColumnByField:](#)

This method returns the column object that matches the specified field name.

```
`ts
```

```
let column = this.grid.getColumnByField('ProductName');
```

```
`
```

- [getColumnByUid:](#)

This method returns the column object that matches the specified UID.

```
`ts
```

```
let column = this.grid.getColumnByUid();
```

```
`
```

- [getVisibleColumns:](#)

This method returns the array of visible columns.

```
`ts
```

```
let visibleColumns = this.grid.getVisibleColumns();
```

```
`
```

- [getForeignKeyColumns:](#)

This method returns the array of foreignkey columns.

```
`ts
```

```
let foreignKeyColumns = this.grid.getForeignKeyColumns();
```

```
`
```

- [getColumnFieldNames](#)

This method returns an array of field names of all the columns in the Grid.

```
`ts
```

```
let fieldNames = this.grid.getColumnFieldNames()
```

```
`
```

For a complete list of column methods and properties, refer to this [section](#).

#### *Updating column definitions*

You can update the column definitions in the Grid using the [columns](#) property. You can modify the properties of the column objects in the columns array to update the columns dynamically. For example, you can change the headerText, width, visible, and other properties of a column to update its

appearance and behavior in the grid and then call the [refreshColumns](#) method to apply the changes to the grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button id="btnId" cssClass="e-info"
(click)="updateColumns()"> Update Columns </button>
    <ejs-grid #grid [dataSource]='data'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  updateColumns(): void {
    // Modifying column properties
    (this.grid as GridComponent).columns[0].textAlign = 'Center';
    (this.grid as GridComponent).columns[0].width = '100';
    (this.grid as GridComponent).columns[2].visible = false;
    (this.grid as GridComponent).columns[1].customAttributes = { class:
'customcss' };
    // Applying changes to the grid
    (this.grid as GridComponent).refreshColumns();
  }
}
```

```
}

```

### MAIN.TS

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

### Adding/Removing Columns

The Grid component allows you to dynamically add or remove columns to and from the grid using the [columns](#) property, which can be accessed through the instance of the Grid.

To add a new column to the Grid, you can directly **push** the new column object to the columns property. To remove a column from the Grid, you can use the **pop** method, which removes the last element from the columns array of the Grid. Alternatively, you can use the splice method to remove a specific column from the columns array.

Here's an example of how you can add and remove a column from the grid:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent, Column } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button id='add' cssClass="e-info"
(click)='addColumn()'> Add Column</button>
<button ej2-button id='delete' cssClass="e-info"
(click)='deleteColumns()'> Delete Column</button>
<ejs-grid #grid [dataSource]='data' [height]='280' >
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
    <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
```

```

        <e-column field='ShipCity' headerText='Ship City'
width=120 ></e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
        }
        addColumns(): void {
            var newColumns = [
                { field: 'EmployeeID', headerText: 'EmployeeID', width: 120 },
                { field: 'OrderDate', headerText: 'Order Date', width: 120,
format: 'yMd' },
            ];
            newColumns.forEach((col) => {
                (this.grid as GridComponent).columns.push(col as Column);
            });
            (this.grid as GridComponent).refreshColumns();
        }
        deleteColumns(): void {
            (this.grid as GridComponent).columns.pop();
            (this.grid as GridComponent).refreshColumns();
        }
    }
}

```

## 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 refresh columns

You can use the [refreshColumns](#) method of the Syncfusion Grid to refresh the columns in the grid. This method can be used when you need to update the grid columns dynamically based on user actions or data changes.

```
`ts
```

```
this.grid.refreshColumns();
```

```
`
```

### Responsive columns

The Syncfusion Angular Grid provides a built-in feature to toggle the visibility of columns based on media queries using the [hideAtMedia](#) property of the column object. The `hideAtMedia` accepts valid [Media Queries](#).

In this example, we have a Grid that displays data with three columns: **Order ID**, **Customer ID**, and **Freight**. We have set the `hideAtMedia` property of the **OrderID** column to (min-width: 700px) which means that this column will be hidden when the browser screen width is less than or equal to 700px.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120 hideAtMedia='(min-width: 700px)'>
        </e-column> // column visibility hide when browser
screen width less than 700px;
      <e-column field='CustomerID' headerText='Customer ID'
width=140 hideAtMedia='(max-width: 700px)'>
        </e-column> // column visibility show when browser
screen width 500px or less;
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C' width=120
hideAtMedia='(min-width: 500px)'>
        </e-column> // column visibility hide when browser
screen width less than 500px;
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=140>
        </e-column> // it always shown
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

**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

- [Group Column by Format](#)
- [Format grid with auto generation columns in Angular Grid](#)
- [Background color change for stacked headers and calculated columns in Angular Grid](#)

- [Drag and Drop Between two grids in Angular Grid](#)

## Row in Angular Grid component

Each row typically represents a single record or item from a data source. Rows in a grid are used to present data in a tabular format. Each row displays a set of values representing the fields of an individual data record. Rows allow users to interact with the data in the grid. Users can select rows, edit cell values, perform sorting or filtering operations, and trigger events based on actions.

### Customize row styles

Customizing the styles of rows in a Syncfusion Grid allows you to modify the appearance of rows to meet your design requirements. This feature is useful when you want to highlight certain rows or change the font style, background color, and other properties of the row to enhance the visual appeal of the grid. To customize the row styles in the grid, you can use CSS, properties, methods, or event support provided by the Syncfusion Angular Grid component.

### Using event

You can customize the appearance of the rows by using the [rowDataBound](#) event. This event triggers for every row when it is bound to the data source. In the event handler, you can get the [RowDataBoundEventArgs](#) object, which contains details of the row. You can use this object to modify the row's appearance, add custom elements, or perform any other customization.

Here's an example of how you can use the `rowDataBound` event to customize the styles of rows based on the value of the **Freight** column. This example involves checking the value of the Freight column for each row and adding a CSS class to the row based on the value. The CSS classes **below-30**, **below-80**, and **above-80** can then be defined in your stylesheet to apply the desired styles to the rows.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data , columnDataType} from './datasource';
import { RowDataBoundEventArgs } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [enableHover]='false'
    [allowSelection]='false' [height]='315'
    (rowDataBound)='rowDataBound($event)'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=120>
      </e-column>
    </e-columns>
  `
})
export class AppComponent {
  constructor() {}
}
```

```

        <e-column field='Freight' headerText='Freight'
textAlign='Right'
        format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=130 >
            </e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
      this.data = data;
    }
    rowDataBound(args: RowDataBoundEventArgs) {
      const Freight = 'Freight';
      if ((args.data as columnDataType)[Freight] < 30) {
        (args.row as Element).classList.add('below-30');
      } else if ((args.data as columnDataType)[Freight] >= 30 &&
((args.data as columnDataType)[Freight] < 80)) {
        (args.row as Element).classList.add('below-80');
      } else {
        (args.row as Element).classList.add('above-80');
      }
    }
  }
}

```

## MAIN.TS

```

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

```

The [queryCellInfo](#) event can also be used to customize cells and is triggered for every cell in the grid. It can be useful when you need to customize cells based on certain conditions or criteria.

### Using CSS

You can apply styles to the rows using CSS selectors. The Grid provides a class name for each row element, which you can use to apply styles to that specific row.

### Customize alternate rows

You can customize the appearance of the alternate rows using CSS. This can be useful for improving the readability of the data and making it easier to distinguish between rows. By default, Syncfusion Grid provides the CSS class **.e-altrow** to style the alternate rows. You can customize this default style by overriding the **.e-altrow** class with your custom CSS styles.

To change the background color of the alternate rows, you can add the following CSS code to your application's stylesheet:

```

`css
.e-grid .e-altrow {
background-color: #fafafa;

```

```
}
,
```

Here's an example of how to use the **.e-altrow** class to style alternate rows:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=140>
      </e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right'
format='C' width=120></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right'
format='yMd' width=140></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = 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 selected row

You can customize the appearance of the selected row using CSS. This is useful when you want to highlight the currently selected row for improve the visual appeal of the Grid. By default, the Grid



provides the CSS class **.e-selectionbackground** to style the selected row. You can customize this default style by overriding the **.e-selectionbackground** class with your own custom CSS styles.

To change the background color of the selected row, you can add the following CSS code to your application:

```
`css
.e-grid .e-selectionbackground {
background-color: #f9920b;
}
```

Here's an example of how to use the **.e-selectionbackground** class to style the selected row:

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=140>
      </e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right'
format='C' width=120></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right'
format='yMd' width=140></e-column>
    </e-columns>
  </ejs-grid>`,
  styles: [`.e-grid .e-selectionbackground {
background-color: #f9920b;
}`]
})
export class AppComponent implements OnInit {

  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

```
}
}
```

### MAIN.TS

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

### Using method

The Grid provides below methods to customize the appearance of the grid rows :

1. [getRowByIndex](#): This method returns the HTML element of a row at the specified index. You can use this method to apply custom styles to a specific row.
2. [getRowIndexByPrimaryKey](#): This method returns the index of the row with the specified primary key. You can use this method to get the index of a specific row and then apply custom styles to it.
3. [getRows](#): This method returns an array of all the row elements in the Grid. You can use this method to apply custom styles to all rows or to a specific set of rows based on some condition.
4. [getRowInfo](#): This method returns the data object and index of the row corresponding to the specified row element. You can use this method to apply custom styles based on the data in a row.
5. [getSelectedRowIndexes](#): This method returns an array of the indexes of the selected rows in the Grid. You can use this method to apply custom styles to the selected rows.
6. [getSelectedRows](#): This method returns an array of the HTML elements representing the selected rows in the grid. You can use this method to directly loop through the selected rows and customize their styles.

The following example demonstrates how to use [getRowByIndex](#) methods to customize the appearance of the row inside the [dataBound](#) event of the grid.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]="data"
(dataBound)="customizeRows()">
    <e-columns>
      <e-column field="OrderID" headerText="Order ID"
textAlign="Right"

```

```

width="100"></e-column>
<e-column field="CustomerID" headerText="Customer ID"
width="120">
</e-column>
<e-column field="Freight" headerText="Freight"
textAlign="Right"
format="C" width="100">
</e-column>
<e-column field="OrderDate" headerText="Order Date"
textAlign="Right"
format="yMd" width="100"></e-column>
</e-columns>
</ejs-grid>`,
}))
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  public customizeRows() {
    ((this.grid as GridComponent).getRowByIndex(2) as
HTMLElement).style.background = 'rgb(193, 228, 234)';
  }
}

```

### 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

The Syncfusion Grid allows you to customize the height of rows based on your needs. This feature can be useful when you need to display more content in a row or when you want to reduce the height of rows to fit its content. You can achieve this by using the [rowHeight](#) property of the Grid component. This property allows you to change the height of the entire grid row to your desired value.

In the below example, we will demonstrate how to dynamically change the height of the rows using the `rowHeight` property.

### APP.COMPONENT.TS

```

import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { orderDetails } from './datasource';
@Component({
  imports: [ GridModule, FormsModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>

```

```

        <button ej-button id="small" cssClass="e-small"
(click)="clickHandler($event)">
            Change height 20px</button>
        <button ej-button id="medium" cssClass="e-small"
(click)="clickHandler($event)">
            Default height 42px</button>
        <button ej-button id="big" cssClass="e-small"
(click)="clickHandler($event)">
            Change height 60px</button>
    </div>
    <div class="control-section" style="padding-top:20px">
        <ejs-grid #grid [dataSource]='data' rowHeight='42' height='400'
[toolbar]='toolbar'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
width='120' textAlign='Right'>
            </e-column>
                <e-column field='CustomerName' headerText='Customer Name'
width='150'>
            </e-column>
                <e-column field='OrderDate' headerText='Order Date'
width='130' format='yMd'
            <e-column field='Freight' headerText='Freight' width='120'
format='C2'
            <e-column field='ShippedDate' headerText='Shipped Date'
width='140' format='yMd'
            <e-column field='ShipCountry' headerText='Ship Country'
width='150'></e-column>
            </e-columns>
        </ejs-grid>
    </div>`
    })
    export class AppComponent {
        public data?: Object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        public toolbar?: Object[];
        public heightRow: { [key: string]: number } = {
            small: 20,
            medium: 40,
            big: 60
        };
        ngOnInit(): void {
            this.data = orderDetails;
        }
        clickHandler(args: MouseEvent): void {
            (this.grid as GridComponent).rowHeight = this.heightRow[(args.target as
HTMLInputElement).id];
        }
    }
}

```

**MAIN.TS**

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

- \* The **rowHeight** property can only be used to set the height of the entire grid row. It cannot be used to set the height of individual cells within a row.
- \* The **rowHeight** property applies the height to all rows in the grid, including the header and footer rows.
- \* You can also set the height for a specific row using the **rowHeight** property of the corresponding row object in the **rowDataBound** event.

#### *Customize row height for particular row*

Customizing the row height for a particular row can be useful when you want to display more content in a particular row, reduce the height of a row to fit its content, or make a specific row stand out from the other rows in the grid. This can be achieved by using the **rowHeight** property of the Grid component along with the **rowDataBound** event.

The **rowHeight** property of the Grid component allows you to set the height of all rows in the grid to a specific value. However, if you want to customize the row height for a specific row based on the row data, you can use the **rowDataBound** event. This event is triggered every time a request is made to access row information, element, or data, and before the row element is appended to the Grid element.

In the below example, the row height for the row with **OrderID** as '10249' is set as '90px' using the **rowDataBound** event.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data , columnDataType} from './datasource';
import { RowDataBoundEventArgs } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [height]='315'
    (rowDataBound)='rowDataBound($event)'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=120>
          </e-column>
```

```

        <e-column field='Freight' headerText='Freight'
textAlign='Right'
        format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=130 >
            </e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
      this.data = data;
    }
    public rowDataBound(args: RowDataBoundEventArgs) {
      if ((args.data as columnDataType) ['OrderID'] === 10249) {
        args.rowHeight = 90;
      }
    }
  }
}

```

### MAIN.TS

```

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

```

\* In virtual scrolling mode, it is not applicable to set different row heights.

\* You can customize the row height of multiple rows by checking the relevant criteria in the `rowDataBound` event and setting the `rowHeight` property accordingly.

\* In the `rowDataBound` event handler, you can access the current row using the `args.row` property and set the `rowHeight` property for that row using the `setAttribute` method.

### Row hover

The Row Hover feature in Grid provides a visual effect when the mouse pointer hovers over the rows, making it easy to highlight and identify the selected row. This feature can also improve the readability of data in the grid. The row hover effect can be enabled or disabled using the `enableHover` property of the Grid component.

By default, the `enableHover` property is set to **true**, which means that the row hovering effect is enabled. To disable the row hover effect, set the `enableHover` property to **false**.

Here is an example that demonstrates how to enable/disable row hover based on the Switch Component:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { SwitchModule } from '@syncfusion/ej2-angular-buttons'

```

```

import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    GridModule,
    SwitchModule
  ],
  providers: [DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<div style="padding:0px 0px 20px 0px">
    <label> Enable/Disable Row Hover</label>
    <ejs-switch id="switch" [checked]="true"
      (change)="toggleRowHover($event)"></ejs-switch>
    </div>
    <ejs-grid #grid [dataSource]='data'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150>
          </e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  toggleRowHover(args: CustomChangeEventArgs): void {
    (this.grid as GridComponent).enableHover = args.checked;
  }
}
interface CustomChangeEventArgs extends ChangeEventArgs {
  checked: boolean;
}

```

## MAIN.TS

```

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

```

The `enableHover` property applies to the entire grid, not individual rows or columns.

#### *How to get the row information when hovering over the cell*

You can retrieve row information when hovering over a specific cell. This can be useful if you want to display additional details or perform some action based on the data in the row. This can be achieved by using the `rowDataBound` event and the `getRowInfo` method of the Grid.

- The `rowDataBound` event is triggered every time a request is made to access row information, element, or data, before the row element is appended to the Grid element.
- The `getRowInfo` method is used to retrieve the row information when hovering over a specific cell. This method takes a single parameter, which is the target element that is being hovered over.

Here's an example that demonstrates how to use the `rowDataBound` event and `getRowInfo` method to retrieve the row information when hovering over a cell in the Syncfusion Grid.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { TooltipModule } from '@syncfusion/ej2-angular-popups'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent, RowDataBoundEventArgs } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
    TooltipModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id='show' style="padding:0px 0px 20px 0px;" ></div>
    <ejs-grid #grid [dataSource]='data'
    (rowDataBound)='rowDataBound($event)'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right'
        width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=140>
        </e-column>
        <e-column field='Freight' headerText='Freight'
        textAlign='Right'
        format='C' width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        textAlign='Right'
        format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>`,
})
export class AppComponent implements OnInit {
  @ViewChild('grid', { static: true }) public grid?: GridComponent;
```



```

public data?: object[];
ngOnInit(): void {
    this.data = data;
}
rowDataBound(args: RowDataBoundEventArgs): void {
    (args.row as HTMLElement).addEventListener('mouseover', (e: MouseEvent)
=> {
        const rowInformation = (this.grid as
GridComponent).getRowInfo(e.target as HTMLElement);
        console.log(rowInformation);
        (document.getElementById('show') as HTMLElement).innerHTML = `
        <table style="border-collapse: collapse; width: 600px;">
            <tr style="border: 2px solid;">
                <td style="padding: 2px;"><b>Row Information:</b></td>
            </tr>
            <tr style="border: 2px solid; padding: 8px;">
                <th style="border: 2px solid; padding: 8px; width:
120px;"><b>Class Name</b>
                </th>
                <td style="border: 2px solid; padding:
8px;">${(rowInformation.row as Element).className}
                </td>
            </tr>
            <tr style="border: 2px solid;">
                <th style="border: 2px solid; padding: 8px;"><b>Row Index</b>
                </th>
                <td style="border: 2px solid; padding:
8px;">${rowInformation.rowIndex}
                </td>
            </tr>
        </table>`;
    });
}
}

```

## MAIN.TS

```

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

```

The `getRowInfo` method can only be used in the `rowDataBound` event. Attempting to use it elsewhere will result in an error.

## Row pinning (Frozen)

The Syncfusion Angular Grid allows you to freeze rows to keep them visible while scrolling vertically through large datasets. This feature enhances the experience by maintaining important information within view at all times.

In the following example, the `frozenRows` property is set to **2**. This configuration freezes the top three rows of the grid, and they will remain fixed in their positions while the rest of the grid can be scrolled vertically.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FreezeService, SelectionService, EditService,
ToolbarService } from '@syncfusion/ej2-angular-grids'
import { NumericTextBoxAllModule, RatingAllModule } from '@syncfusion/ej2-
angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent, } from '@syncfusion/ej2-angular-grids';
import { NumericTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { data } from './datasource';
@Component({
imports: [

    GridModule,
    NumericTextBoxAllModule,
    RatingAllModule,
    ButtonModule

],
providers: [FreezeService, SelectionService, EditService, ToolbarService],
standalone: true,
selector: 'app-root',
template: `<div style="display: flex">
    <label style="padding: 10px 10px 26px 0"> Change the frozen rows:
</label>
    <ejs-numerictextbox
        #frozenRows
        min="0"
        max="5"
        [validateDecimalOnType]="true"
        decimals="0"
        format="n"
        value="2"
        width="100px"
    ></ejs-numerictextbox>
    <div>
        <button style="margin-left:5px" ej-button (click)="frozenRowsFn()">
            Update
        </button>
    </div>
</div>
    <ejs-grid #grid style="padding: 5px 5px" [dataSource]='data' height=315
[frozenRows]='2' [allowSelection]='false' [enableHover]='false'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID' width=100></e-
column>
        <e-column field='OrderDate' headerText='Order Date' width=100
format='yMd' textAlign='Right'></e-column>
        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=80></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=130></e-
column>
        <e-column field='ShipAddress' headerText='Ship Address' width=140></e-
column>

```

```

        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country' width=100></e-
column>
        <e-column field='ShipRegion' headerText='Ship Region' width=80></e-
column>
        <e-column field='ShipPostalCode' headerText='Ship Postal Code'
width=110></e-column>
        <e-column field='Freight' headerText='Freight' width=80></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        @ViewChild('frozenRows')
        public frozenRows?: NumericTextBoxComponent;
        ngOnInit(): void {
            this.data = data
        }
        frozenRowsFn() {
            (this.grid as GridComponent).frozenRows = (this.frozenRows as
NumericTextBoxComponent).value;
        }
    }
}

```

## MAIN.TS

```

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

```

- \* Frozen rows should not be set outside the grid view port.
- \* Frozen Grid will support row virtualization feature, which helps to improve the Grid performance while loading a large dataset.
- \* The frozen feature is supported only for the rows that are visible in the current view.
- \* You can use both [frozenColumns](#) property and `frozenRows` property in the same application.

### *Change default frozen rows line color*

You can easily customize the frozen line background color of frozen rows in the Syncfusion Grid component by applying custom CSS styles to the specific frozen row. This allows you to change the background color of frozen rows to match your application's design and theme.

To change the default frozen rows line color, you can use the following CSS class:

```

`css
.e-grid .e-frozenrow-border {
background-color: rgb(5, 114, 47);
}

```

By applying this CSS class, you can set the background color of frozen rows to the specified RGB color. The following example demonstrates how to change the default frozen rows line color using CSS.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FreezeService, SelectionService, EditService,
ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule,
  ],
  providers: [FreezeService, SelectionService, EditService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height=315
[allowSelection]='false' [frozenRows]='3' [enableHover]='false'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=100 ></e-
column>
    <e-column field='OrderDate' headerText='Order Date' width=100
format='yMd' textAlign='Right'></e-column>
    <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=80></e-column>
    <e-column field='ShipName' headerText='Ship Name' width=130></e-
column>
    <e-column field='ShipAddress' headerText='Ship Address' width=140
></e-column>
    <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
    <e-column field='ShipCountry' headerText='Ship Country' width=100></e-
column>
    <e-column field='ShipRegion' headerText='Ship Region' width=80></e-
column>
    <e-column field='ShipPostalCode' headerText='Ship Postal Code'
width=110></e-column>
    <e-column field='Freight' headerText='Freight' width=80></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];

  ngOnInit(): void {
    this.data = data
  }
}
```

**MAIN.TS**

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

*Deprecated methods*

Previous | Current | Explanation

**getMovableRows()** `gridInstance.getMovableRows()[0].querySelectorAll('.e-unfreeze')` | **getRows()** `gridInstance.getRows()[0].querySelectorAll('.e-unfreeze')` | The previous architecture used separate tables for left, right, and movable contents, returning only movable rows when calling the method, whereas the current architecture combines them into one table, returning all rows and introduces the **e-unfreeze** class for selecting movable rows

**getFrozenRightRows()** `gridInstance.getFrozenRightRows()[0].querySelectorAll('.e-rightfreeze')` | **getRows()** `gridInstance.getRows()[0].querySelectorAll('.e-rightfreeze')` | In the previous architecture, it returned only the table rows from the right freeze table, but in the current architecture, all rows of the entire table are returned, introducing the **e-rightfreeze** class for selecting right freeze rows.

**getMovableRowByIndex()** <br> **getFrozenRowByIndex()** <br> **getFrozenRightRowByIndex()** | **getRowByIndex()** `gridInstance.getRowByIndex(1).querySelectorAll('.e-unfreeze')` | In the previous architecture, separate methods were used to select rows from different table sections, while in the current architecture, the **getMovableRowByIndex()**, **getFrozenRightRowByIndex()**, and **getFrozenRowByIndex()** methods now return the same table row based on the given index. Additionally, class names for table cells (td's) have been separated into **e-leftfreeze**, **e-unfreeze**, and **e-rightfreeze**, making it easier to customize cells within a row.

**getMovableCellFromIndex()** <br> **getFrozenRightCellFromIndex()** | **getCellFromIndex()** `gridInstance.getCellFromIndex(1,1)` | In the previous approach, the **getMovableCellFromIndex()** method was used to choose a specific cell within the movable table, and the **getFrozenRightCellFromIndex()** method was utilized to target a particular cell within the right freeze table. However, in the current architecture, you have the flexibility to select a specific cell in either the movable or right freeze table by using both the **getFrozenRightCellFromIndex()** and **getMovableCellFromIndex()** methods. This new method simplifies the process of selecting and retrieving specific cells within these tables, offering more versatility and convenience.

**getMovableDataRows()** <br> **getFrozenRightDataRows()** <br> **getFrozenDataRows()** | **getDataRows()** `gridInstance.getDataRows()[0].querySelectorAll('.e-unfreeze')` | In the previous approach, there were separate methods (**getMovableDataRows()**, **getFrozenRightDataRows()**, and **getFrozenDataRows()**) for obtaining viewport data rows from the freeze, movable, and right tables individually. However, in the new approach, these methods have been enhanced to return the entire viewport data rows for all sections together, simplifying data retrieval. You can now extract specific cells within these rows using selectors such as **e-leftfreeze** for the **left freeze**, **e-unfreeze** for the **movable**, and **e-rightfreeze** for the **right freeze** tables, providing greater flexibility in working with the data.

**getMovableColumnHeaderByIndex()** <br> **getFrozenRightColumnHeaderByIndex()** <br> **getFrozenLeftColumnHeaderByIndex()** | **getColumnHeaderByIndex()** `gridInstance.getColumnHeaderByIndex(1)` | In the previous architecture, the methods selected movable,

right freeze, and left freeze headers separately. However, in the new approach, when using the `getMovableColumnHeaderByIndex()`, `getFrozenRightColumnHeaderByIndex()`, and `getFrozenLeftColumnHeaderByIndex()` methods, you will still obtain the same results as in the previous architecture.

When a validation message is displayed in the frozen part (Left, Right, Fixed) of the table, scrolling is prevented until the validation message is cleared.

#### Limitations

- Frozen row is not compatible with the following features:
  1. Autofill

#### Adding a new row programmatically

The Syncfusion Grid provides a way to add a new row to the grid programmatically. This feature is useful when you want to add a new record to the grid without having the manually enter data in the grid. This can be done using the [addRecord](#) method of the Grid.

The `addRecord` method takes two parameters:

- The **data** object representing the new row to be added
- The **index** at which the new row should be inserted. If no index is specified, the new row will be added at the end of the Grid.

Here's an example of how to add a new row using the `addRecord` method:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService } from '@syncfusion/ej2-angular-grids'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { EditSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    ButtonModule
  ],
  providers: [EditService],
  standalone: true,
  selector: 'app-root',
  template: `<div style="padding:0px 0px 20px 0px">
```

```

        <button ej-button id='add' (click)='addRow()'>Add New
Row</button>
    </div>
    <ejs-grid #grid id="grid" [dataSource]='data'
[editSettings]='editSettings'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=100 isPrimaryKey="true"></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' width=120>
            </e-column>
            <e-column field='ShipCity' headerText='ShipCity'
width=100>
            </e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right'
format='C' width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=150>
            </e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public editSettings?: EditSettingsModel;
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data; // Initialize an empty array
            this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
        }
        addRow() {
            const newRecord = {
                OrderID: this.generateOrderId(),
                CustomerID: this.generateCustomerId(),
                ShipCity: this.generateShipCity(),
                Freight: this.generateFreight(),
                ShipName: this.generateShipName()
            };
            (this.grid as GridComponent).addRecord(newRecord, 0);
        }
        // Generate a random OrderID
        generateOrderId(): number {
            return Math.floor(10000 + Math.random() * 90000);
        }
        // Generate a random CustomerID
        generateCustomerId(): string {
            const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
            let result = '';
            for (let i = 0; i < 5; i++) {
                result += characters.charAt(Math.floor(Math.random() *
characters.length));
            }
            return result;
        }
    }

```

```

    }
    // Generate a random ShipCity
    generateShipCity(): string {
        const cities = ['London', 'Paris', 'New York', 'Tokyo', 'Berlin'];
        return cities[Math.floor(Math.random() * cities.length)];
    }
    // Generate a random Freight value
    generateFreight(): number {
        return Math.random() * 100;
    }
    // Generate a random ShipName
    generateShipName(): string {
        const names = ['Que Delicia', 'Bueno Foods', 'Island Trading',
        'Laughing Bacchus Winecellars'];
        return names[Math.floor(Math.random() * names.length)];
    }
}

```

## MAIN.TS

```

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

```

- \* When working with remote data, it is impossible to add a new row between the existing rows.
- \* If you want to add a new record to the beginning of the data source, you can pass **0** as the second parameter to the `addRecord` method.
- \* If you do not specify an index, the new row will be added at the end of the grid.

### Show or hide a row using an external actions

In a Syncfusion grid, you can show or hide a particular row based on some external action, such as a checkbox click. This can be useful in scenarios where you want to hide certain rows from the grid temporarily, without removing them from the underlying data source. This can be achieved by using the `getRowByIndex` and `getRowsObject` methods of the grid along with the `change` event of the checkbox.

The `getRowsObject` method returns an array of row objects that represents all the rows in the grid. You can use this method to iterate through all the rows and access their data and index.

The `getRowByIndex` method returns the HTML element of a row at the specified index. You can use this method to get a specific row and apply changes to it.

In the following example, the `onCheckBoxChange` method is used to check whether the checkbox is checked or not. If it is checked, the method iterates through all the rows in the grid using the `getRowsObject` method. For each row, it checks whether the value in the **CustomerID** column is equal to "VINET". If it is, the index of that row is obtained using the `getRowByIndex` method and hidden by setting its display style to "none". The index of the hidden row is also added to an array called `hiddenRows`.

If the checkbox is unchecked, the method iterates through the `hiddenRows` array and shows each row by setting its display style to an empty string. The `hiddenRows` array is also cleared.



**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { orderDetails, columnDataType } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    GridModule,
    CheckBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div style="padding:2px 0px 0px 0px">
    <ejs-checkbox #checkbox label='Show / Hide Row'
      (change)="onCheckBoxChange($event)"></ejs-checkbox>
  </div>
  <p id="message">{{ message }}</p>
  <ejs-grid #grid [dataSource]='data' height='350'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
width='120'
      textAlign='Right'></e-column>
      <e-column field='CustomerID' headerText='Customer Name'
width='150'>
      </e-column>
      <e-column field='OrderDate' headerText='Order Date'
width='130'
      format="yMd" textAlign='Right'>
      </e-column>
      <e-column field='Freight' headerText='Freight'
width='120' format='C2'
      textAlign='Right'>
      </e-column>
      <e-column field='ShippedDate' headerText='Shipped Date'
width='130'
      format="yMd" textAlign='Right'>
      </e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
width='150'>
      </e-column>
    </e-columns>
  </ejs-grid>`,
  })
export class AppComponent {
  public data: Object[] = [];
  public rowIndex?: number;
  public hiddenRows: number[] = [];
  @ViewChild('grid')
  public grid?: GridComponent;
  public message?: string = '';
  ngOnInit(): void {

```

```

    this.data = orderDetails;
  }
  public onCheckBoxChange(args: ChangeEventArgs) {
    if (args.checked) {
      for (let i = 0; i < (this.grid as
GridComponent).getRowsObject().length; i++) {
        if (((this.grid as GridComponent).getRowsObject()[i].data as
columnDataType).CustomerID === 'VINET') {
          // check the row value
          this.rowIndex = (this.grid as
GridComponent).getRowsObject()[i].index; //get particular row index
          ((this.grid as GridComponent).getRowByIndex(this.rowIndex) as
HTMLElement).style.display =
            'none'; //hide row
          this.hiddenRows.push((this.rowIndex as number)); // add row index
to hiddenRows array
        }
      }
      if (this.hiddenRows.length > 0) {
        this.message = `Rows with a customer name column value of VINET have
been hidden`;
      }
    } else {
      // Show hidden rows
      this.hiddenRows.forEach((rowIndex: number) => {
        ((this.grid as GridComponent).getRowByIndex(rowIndex) as
HTMLElement).style.display = '';
      });
      this.hiddenRows = [];
      this.message = 'Show all hidden rows';
    }
  }
}

```

### 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 get the row data and element

Grid provides several methods to retrieve row data and elements. This feature is useful when you need to access specific rows, perform custom operations, or manipulate the data displayed in the grid.

1. [getRowByIndex](#): This method returns the HTML element of a row at the specified index. It can be used to retrieve the element of a specific row in the grid.

`ts

```
const rowElement = this.grid.getRowByIndex(rowIndex);
```

,

2. [getRowIndexByPrimarykey](#):The method allows you to retrieve the row index based on a specific primary key value or row data.

```
`ts
```

```
const rowIndex = this.grid.getRowIndexByPrimarykey(primarykey);
```

```
,
```

3. [getRowInfo](#):This method allows you to retrieve row information based on a cell target element.

```
`ts
```

```
const rowInformation = this.grid.getRowInfo(targetElement);
```

```
,
```

4. [getRows](#): This method returns an array of all the row elements in the Grid. If you need to retrieve row data and elements, you can combine the `getRows` method with the `getRowInfo` method.

```
`ts
```

```
const rowElements = this.grid.getRows();
```

```
,
```

5. [getSelectedRowIndex](#):This method allows you to retrieve the collection of indexes of the selected rows. However, it does not directly provide the row elements and associated data. To access the row elements and data of the selected rows, you can combine the `getSelectedRowIndex` method with `getRowByIndex` and `getRowInfo` method.

```
`ts
```

```
const selectedIndexes = this.grid.getSelectedRowIndex();
```

```
,
```

6. [getSelectedRows](#):This method returns an array of HTML elements representing the selected rows in the grid.By iterating over this array, you can access each row element and data using the `getRowInfo` method. This way, you can access both the row elements and their associated data for the selected rows.

```
`ts
```

```
const selectedRowElements = this.grid.getSelectedRows();
```

```
,
```

See Also

- [How to customize the row height in Angular Grid](#)
- [How to set font size and padding of Angular Grid's toolbar and filter bar](#)

- [How to displaying serial number in Angular Grid](#)
- [How to add/update a new row programmatically in Angular Grid](#)

### Cell in Angular Grid component

In the Syncfusion Angular Grid, a **cell** refers to an individual data point or a unit within a grid column that displays data. It represents the intersection of a row and a column, and it contains specific information associated with that row and column. Each cell can display text, numbers, or other content related to the data it represents.

The Grid component allows you to customize the appearance and behavior of cells using various features and options. You can define templates, format cell values, enable or disable editing, and perform various other operations on the cells to create interactive and informative data grids in your web applications.

### Displaying the HTML content

Displaying HTML content in a Grid can be useful in scenarios where you want to display formatted content, such as images, links, or tables, in a tabular format. Grid component allows you to display HTML tags in the Grid header and content. By default, the HTML content is encoded to prevent potential security vulnerabilities. However, you can enable the [disableHtmlEncode](#) property by setting the value as false to display HTML tags without encoding. This feature is useful when you want to display HTML content in a grid cell.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to enable and disable the [disableHtmlEncode](#) property. When the switch is toggled, the [change](#) event is triggered and the [disableHtmlEncode](#) property of the column is updated accordingly. The [refreshColumns](#) method is called to refresh the grid and display the updated content.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import {
  ButtonModule,
  CheckBoxModule,
  RadioButtonModule,
  SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GridModule,
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
  ],
  providers: [DetailRowService],
  standalone: true,
```

```

selector: 'app-root',
template: `
<div>
<label style="padding: 10px 10px">
Enable or disable HTML Encode
</label>
<ejs-switch id="switch" (change)="change($event)"></ejs-switch>
</div>
<ejs-grid #grid [dataSource]='data' [height]='315' style="padding: 10px
10px">
  <e-columns>
    <e-column field='OrderID' headerText= 'Order ID'
textAlign='Right' width=140></e-column>
    <e-column field='CustomerID' headerText="<strong> Customer ID
</strong>" width=120></e-column>
    <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=80></e-column>
    <e-column field='ShipCity' headerText='Ship City' width=130
></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  change(args: ChangeEventArgs) {
    if (args.checked) {
      (this.grid as
GridComponent).getColumnByField('CustomerID').disableHtmlEncode = false;
    } else {
      (this.grid as
GridComponent).getColumnByField('CustomerID').disableHtmlEncode = true;
    }
    (this.grid as GridComponent).refreshColumns();
  }
  ngOnInit(): void {
    this.data = data;
  }
}

```

## MAIN.TS

```

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

```

- \* The [disableHtmlEncode](#) property disables HTML encoding for the corresponding column in the grid.
- \* If the property is set to **true**, any HTML tags in the column's data will be displayed.
- \* If the property is set to **false**, the HTML tags will be removed and displayed as plain text.
- \* Disabling HTML encoding can potentially introduce security vulnerabilities, so use caution when enabling this feature.

\* If [enableHtmlSanitizer](#) property of grid is set to true, then the content is sanitized to prevent any potential security vulnerabilities.

\* You can also disable the `disableHtmlEncode` property of the column using [getColumns](#) method on [change](#) event of Switch component. This is demonstrated in the below code snippet,

```
`typescript
change(args) {
  if (args.checked) {
    this.grid.getColumns()[1].disableHtmlEncode = false;
  } else {
    this.grid.getColumns()[1].disableHtmlEncode = true;
  }
  this.grid.refresh();
}
```

### Autowrap the grid content

The auto wrap feature allows the cell content in the grid to wrap to the next line when it exceeds the boundary of the specified cell width. The cell content wrapping works based on the position of white space between words. To support the Autowrap functionality in Syncfusion Grid, you should set the appropriate [width](#) for the columns. The column width defines the maximum width of a column and helps to wrap the content automatically.

To enable auto wrap, set the [allowTextWrap](#) property to **true**. You can also configure the wrap mode by setting the [textWrapSettings.wrapMode](#) property.

Grid provides the below three options for configuring:

- **Both** - This is the default value for wrapMode. With this option, both the grid **Header** and **Content** text is wrapped.
- **Header** - With this option, only the grid header text is wrapped.
- **Content** - With this option, only the grid content is wrapped.

The following example demonstrates how to set the `allowTextWrap` property to **true** and specify the wrap mode as **Content** by setting the `textWrapSettings.wrapMode` property. Also change the `textWrapSettings.wrapMode` property to **Content** and **Both** on changing the dropdown value using the [change](#) event of the DropDownList component.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { inventoryData } from './datasource';
```

```

import { GridComponent, TextWrapSettingsModel, WrapMode } from
'@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [

    GridModule,
    DropDownListAllModule
  ],
  providers: [PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0"> Change the wrapmode of auto
wrap feature: </label>
      <ejs-dropdownlist
        style="margin-top: 5px"
        index="0"
        width="100"
        [dataSource]="ddlData"
        (change)="valueChange($event)">
      </ejs-dropdownlist>
    </div>
    <ejs-grid #grid style="padding: 5px 5px" [dataSource]='data'
allowPaging='true' allowTextWrap='true' [textWrapSettings]='wrapSettings'
height='400'>
      <e-columns>
        <e-column field='Inventor' headerText='Inventor Name'
width='180' textAlign="Right"></e-column>
        <e-column field='NumberofPatentFamilies' headerText="Number of
Patent Families" width='180' textAlign="Right"></e-column>
        <e-column field='Country' headerText='Country' width='140'></e-
column>
        <e-column field='Active' width='120'></e-column>
        <e-column field='Mainfieldsofinvention' headerText='Main fields
of invention' width='200'></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public wrapSettings?: TextWrapSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  public ddlData: object[] = [
    { text: 'Content', value: 'Content' },
    { text: 'Both', value: 'Both' },
  ];
  ngOnInit(): void {
    this.data = inventoryData;
    this.wrapSettings = { wrapMode: 'Content' };
  }
  valueChange(args: ChangeEventArgs): void {
    (this.grid as GridComponent).textWrapSettings.wrapMode = args.value as
WrapMode;
  }
}

```

```
}

```

## MAIN.TS

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

\* If a column width is not specified, then the Autowrap of columns will be adjusted with respect to the grid's width.

\* If a column's header text contains no white space, the text may not be wrapped.

\* If the content of a cell contains HTML tags, the Autowrap functionality may not work as expected. In such cases, you can use the [headerTemplate](#) and [template](#) properties of the column to customize the appearance of the header and cell content.

### Customize cell styles

Customizing the grid cell styles allows you to modify the appearance of cells in the Grid control to meet your design requirements. You can customize the font, background color, and other styles of the cells. To customize the cell styles in the grid, you can use grid event, css, property or method support.

#### Using event

To customize the appearance of the grid cell, you can use the [queryCellInfo](#) event of the grid. This event is triggered when each header cell is rendered in the grid, and provides an object that contains information about the header cell. You can use this object to modify the styles of the header cell.

The following example demonstrates how to add a `queryCellInfo` event handler to the grid. In the event handler, checked whether the current column is **Freight** field and then applied the appropriate CSS class to the cell based on its value.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data, columnDataType } from './datasource';
import { QueryCellInfoEventArgs, Column } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [enableHover]='false'
[allowSelection]='false' [height]='315'
(queryCellInfo)='customizeCell($event)'>
<e-columns>
```



```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=130 ></e-column>
    </e-columns>
</ejs-grid>`,
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        ngOnInit(): void {
            this.data = data;
        }
        customizeCell({ data, cell, column }: QueryCellInfoEventArgs) {
            if ((column as Column).field === 'Freight') {
                const freightData = (data as columnDataType).Freight
                if (freightData <= 30) {
                    (cell as Element).classList.add('below-30');
                }
                else if (freightData > 30 && freightData < 80) {
                    (cell as Element).classList.add('below-80');
                } else {
                    (cell as Element).classList.add('above-80');
                }
            }
        }
    }
}

```

## MAIN.TS

```

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

```

\* The [queryCellInfo](#) event is triggered for every cell of the grid, so it may impact the performance of the grid whether used to modify a large number of cells.

### Using CSS

You can apply styles to the cells using CSS selectors. The Grid provides a class name for each cell element, which you can use to apply styles to that specific cell or cells in a particular column. The `e-rowcell` class is used to style the row cells, and the `e-selectionbackground` class is used to change the background color of the selected row.

`CSS

```

.e-grid td.e-cellselectionbackground {
background: #9ac5ee;
font-style: italic;
}

```

The following example demonstrates how to customize the appearance of a specific row in the grid on selection using `className`.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [height]='315'
[selectionSettings]="selectOptions">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=130 ></e-column>
      <e-column field='ShipName' headerText='Ship Name'
textAlign='Right' width=80></e-column>
    </e-columns>
  </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public data: any;
  public selectOptions: any= {
    mode: 'Cell',
    type: 'Multiple',
  };
  ngOnInit(): void {
    this.data = data;
  }
}
```

#### MAIN.TS

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

*Using property*

To customize the style of grid cells, define [customAttributes](#) property to the column definition object. The `customAttributes` property takes an object with the name-value pair to customize the CSS properties for grid cells. You can also set multiple CSS properties to the custom class using the `customAttributes` property.

```
`CSS
.custom-css {
background: #d7f0f4;
font-style: italic;
color:navy
}
```

Here, setting the `customAttributes` property of the **ShipCity** column to an object that contains the CSS class 'custom-css'. This CSS class will be applied to all the cells in the **ShipCity** column of the grid.

```
`typescript
{
field: 'ShipCity', headerText: 'Ship City', customAttributes: {class: 'custom-css'}, width: '120'
}
```

The following example demonstrates how to customize the appearance of the **OrderID** and **ShipCity** columns using custom attributes.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
imports: [

    GridModule
],
providers: [DetailRowService],
standalone: true,
selector: 'app-root',
template: `<ejs-grid [dataSource]='data' [height]='315'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID'
[customAttributes]="{class: 'custom-css'}"
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>`
```

```

        <e-column field='ShipCity' headerText='Ship City'
[customAttributes]="{class: 'custom-css'}" width=130 ></e-column>
        <e-column field='ShipName' headerText='Ship Name'
textAlign='Right' width=80></e-column>
    </e-columns>
</ejs-grid>`
}))
export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
        this.data = data;
    }
}

```

## MAIN.TS

```

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

```

Custom attributes can be used to customize any cell in the grid, including header and footer cells.

### Using methods

The Grid provides below methods to customize the appearance of the grid columns header and cell:

1. [getHeaderContent](#): The `getHeaderContent` method is used to customize the appearance of the column header in the grid and accessing the header element using the `querySelector` method and applying the style using the `style` property of the cell element.
2. [getCellFromIndex](#): The `getCellFromIndex` method is used to customize the appearance of a specific cell in the grid by specifying the index of the row and column for which you want to customize the appearance.

The following example demonstrates how to use [getColumnHeaderByIndex](#) and [getCellFromIndex](#) methods to customize the appearance of the **CustomerID** column header and specific cell inside the [dataBound](#) event of the grid.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [DetailRowService],
    standalone: true,

```

```

    selector: 'app-root',
    template: `<ejs-grid #grid [dataSource]='data' [height]='315'
(dataBound)="dataBound()">
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=130 ></e-column>
            <e-column field='ShipName' headerText='Ship Name'
textAlign='Right' width=80></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    @ViewChild('grid', { static: true })
    public grid?: GridComponent;
    public data?: object[];
    ngOnInit(): void {
      this.data = data;
    }
    dataBound() {
      let header = (this.grid as
GridComponent).getHeaderContent().querySelector('.e-headercell');
      (header as HTMLElement).style.backgroundColor = 'red';
      (header as HTMLElement).style.color = 'white';
      let cell = (this.grid as GridComponent).getCellFromIndex(1, 2);
      (cell as HTMLElement).style.background = '#f9920b';
      (cell as HTMLElement).style.color = 'white';
    }
  }
}

```

## MAIN.TS

```

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

```

Make sure to pass the correct row and column indices to [getCellFromIndex](#) method, or else the appearance of the wrong cell might get customized.

## Clip Mode

The clip mode feature is useful when you have a long text or content in a grid cell, which overflows the cell's width or height. It provides options to display the overflow content by either truncating it, displaying an ellipsis or displaying an ellipsis with a tooltip. You can enable this feature by setting [columns.clipMode](#) property to one of the below available options.

There are three types of [clipMode](#) available:

- **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. Also it will display the tooltip while hover on ellipsis is applied.

The following example demonstrates, how to set the [clipMode](#) property to **Clip**, **Ellipsis** and **EllipsisWithTooltip** for the **Main Fields of Invention** column, on changing the dropdown value using the [change](#) event of the **DropDownList** component. The [refresh](#) method is used to refresh the grid and display the updated content.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { inventoryData } from '../datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { GridComponent, ClipMode } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule, DropDownListAllModule
  ],
  providers: [PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0"> Change the clip mode: </label>
      <ejs-dropdownlist
        style="margin-top: 5px"
        index="0"
        width="150"
        [fields]='fields'
        [dataSource]="ddlData"
        (change)="valueChange($event)"></ejs-dropdownlist>
    </div>
    <ejs-grid #grid style="padding: 5px 5px" [dataSource]='data'
    allowPaging='true'>
      <e-columns>
        <e-column field='MainFieldsofInvention' headerText='Invention'
        width='130'></e-column>
        <e-column field='Inventor' headerText='Inventor' width='80'></e-
        column>
        <e-column field='NumberofPatentFamilies' headerText='Count'
        width='100'></e-column>
        <e-column field='Country' headerText='Country' width='80'></e-
        column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
```

```

public grid?: GridComponent;
public fields: object = { text: 'text', value: 'value' };
public ddlData: object[] = [
  { text: 'Ellipsis', value: 'Ellipsis' },
  { text: 'Clip', value: 'Clip' },
  { text: 'Ellipsis with Tooltip', value: 'EllipsisWithTooltip' },
];
ngOnInit(): void {
  this.data = inventoryData;
}
valueChange(args: ChangeEventArgs): void {
  (this.grid as
GridComponent).getColumnByField('MainFieldsOfInvention').clipMode =
(args.value as ClipMode);
  (this.grid as GridComponent).refresh();
}
}

```

## 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, [columns.clipMode](#) value is **Ellipsis**.

\* If you set the [width](#) property of a column, the clip mode feature will be automatically applied to that column if the content exceeds the specified width.

\* Be careful when using the Clip mode, as it may result in important information being cut off. It is generally recommended to use the Ellipsis or EllipsisWithTooltip modes instead.

## Tooltip

The Syncfusion Grid allows you to display information about the grid columns to the user when they hover over them with the mouse.

### Render bootstrap tooltip in grid cells

The Grid component allows rendering Bootstrap tooltips in the cells. To enable this feature, you need to add the Bootstrap CDN link and use the `ngAfterViewChecked` method to initialize the tooltip.

This is demonstrated in the sample code below which shows how to enable Bootstrap tooltip for the **CustomerID** field using `ng-template` in grid cells,

Step 1: Install the Bootstrap and jQuery package in your application using the following commands and add the script and style of the respective packages in the angular.json file,

To install bootstrap, use the following command.

```
`bash
```

```
npm install @ng-bootstrap/ng-bootstrap
```

```
,
```

To install jQuery, use the following command.

```
`bash
npm install jquery
`
`json
"styles":
[
"node_modules/bootstrap/dist/css/bootstrap.min.css",
"src/styles.css",
"node_modules/@syncfusion/ej2-material-theme/styles/material.css"
],
"scripts": ["node_modules/jquery/dist/jquery.min.js"]
`
```

Step 2: Add the CDN link of Bootstrap in the index.html file. Place the `link` tag in the `head` for the CSS, and the `script` tag for the JavaScript bundle before the closing `body`.

```
`html
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"
crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ENjdO4Dr2bkBIFxQpeoTz1Hlcje39Wm4jDKdf19U8gl4ddQ3GYNS7NTKfAdVQSZ"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.7/dist/umd/popper.min.js"
integrity="sha384-zYPOMqeu1DAVkhILqWBUTcbYfZ8osu1Nd6Z89ify25QV9guujx43ITvfi12/QExE"
crossorigin="anonymous"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/js/bootstrap.min.js"
integrity="sha384-Y4oOpwW3duJdCWv5ly8SCFYWqFDsfob/3GkgExXKV4idmbt98QcxXYs9UoXAB7BZ"
crossorigin="anonymous"></script>
`
```

Step 3: The following code demonstrates how to render Bootstrap tooltip for the **CustomerID** field with `ng-template` on grid cells using `ngAfterViewChecked`` method,

```
`typescript
import { AfterViewChecked, Component } from '@angular/core';
import { orderDataSource } from './data';
import 'bootstrap';
declare var $: any;
```



```

@Component({
  selector: 'app-root',
  template: `
    <div>
      <ejs-grid [dataSource]='data' [allowPaging]='true'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
          <e-column field='CustomerID' headerText='Customer ID' width=120>
            <ng-template #template let-data>
              <div class="row clearfix">
                <div class="col-md-2" style="text-align:right">
                  <div
                    data-toggle="tooltip"
                    data-container="body"
                    [title]="data.CustomerID"
                    data-placement="left"
                    data-delay='{ "show": "800", "hide": "300" }'>
                    <i class="fas fa-pencil-alt"></i>{{ data.CustomerID }}
                  </div>
                </div>
              </div>
            </ng-template>
          </e-column>
          <e-column field='Freight' headerText='Freight' textAlign='Right' format='C2' width=90></e-column>
          <e-column field='OrderDate' headerText='Order Date' textAlign='Right' format='yMd' width=120></e-column>
        </e-columns>
      </ejs-grid>
    </div>`
  })
  export class AppComponent implements AfterViewChecked {
    public data = orderDataSource;
    ngAfterViewChecked() {

```

```

$('[data-toggle="tooltip"]').tooltip();
}
}
,

```

The following screenshot represents the Bootstrap tooltip for the **CustomerID** field,

Order ID	Customer ID	Freight	Order Date
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996
10251	VICTE	\$41.34	7/8/1996
10252	SUPRD	\$51.30	7/9/1996
10253	HANAR	\$58.17	7/10/1996
10254	CHOPS	\$22.98	7/11/1996
10255	RICSU	\$146.33	7/12/1996
10256	WELLI	\$13.97	7/13/1996
10257	HILAA	\$81.91	7/16/1996
10258	ERNSH	\$140.51	7/17/1996
10259	CENTC	\$5.25	7/18/1996

1 of 70 pages (830 items)

\* The Bootstrap CDN link must be added to the HTML file.

\* The `ngAfterViewChecked` method is called after the component's view has been fully initialized, so it is a good place to initialize the Bootstrap tooltip.

#### Display custom tooltip for columns

The Grid provides a feature to display custom tooltips for its columns using the [EJ2 Tooltip](#) component. This allows you to provide additional information about the columns when the user hovers over them.

To enable custom tooltips for columns in the Grid, you can render the Grid control inside the Tooltip component and set the target as `.e-rowcell`. This will display the tooltip when hovering over the grid cells.

Change the tooltip content for the grid cells by using the following code in the [beforeRender](#) event.

```

`typescript
beforeRender(args): void {
if (args.target.classList.contains('e-rowcell')) {
// event triggered before render the tooltip on target element.
this.tooltip.content = 'This is value "' + args.target.innerText + '"';
}
}
,

```

The following example demonstrates how to customize the tooltip content for the grid cells by using the [beforeRender](#) event of the EJ2 Tooltip component.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { TooltipModule } from '@syncfusion/ej2-angular-popups'
import { Component, OnInit, ViewChild } from '@angular/core';
import { TooltipComponent, TooltipEventArgs } from '@syncfusion/ej2-angular-popups';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
    ButtonModule,
    TooltipModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-tooltip #tooltip target=".e-rowcell"
(beforeRender)="beforeRender($event)">
    <ejs-grid [dataSource]='data' [height]='315'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID'
headerText='Customer ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='ShipName' headerText='Ship
Name' width=120></e-column>
      </e-columns>
    </ejs-grid>
  </ejs-tooltip>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('tooltip')
  public tooltip?: TooltipComponent;
  ngOnInit(): void {
    this.data = data;
  }
  beforeRender(args: TooltipEventArgs): void {
    if (args.target.classList.contains('e-rowcell')) {
      (this.tooltip as TooltipComponent).content = 'The value is "' +
args.target.innerText + '"';
    }
  }
}
```

#### 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

The [gridLines](#) in a grid are used to separate the cells with horizontal and vertical lines for better readability. You can enable the grid lines by setting the [gridLines](#) property to one of the following values:

- | 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.|

The following example demonstrates how to set the [gridLines](#) property based on changing the dropdown value using the [change](#) event of the DropDownList component.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { inventoryData } from './datasource';
import { GridComponent, GridLine } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
  providers: [PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0"> Change the grid lines:
    </label>
    <ejs-dropdownlist
      style="margin-top: 5px"
      id="value"
      #dropdown
      index="0"
      width="100"
      [dataSource]="ddlData"
      (change)="valueChange($event)"></ejs-dropdownlist>
```

```

</div>
<ejs-grid #grid style="padding: 5px 5px" [dataSource]='data'
height='315' gridLines='Default'>
  <e-columns>
    <e-column field='Inventor' headerText='Inventor Name'
width='180' textAlign="Right"></e-column>
    <e-column field='NumberofPatentFamilies' headerText="Number of
Patent Families" width='180' textAlign="Right"></e-column>
    <e-column field='Country' headerText='Country' width='140'></e-
column>
    <e-column field='Active' width='120'></e-column>
    <e-column field='Mainfieldsofinvention' headerText='Main fields
of invention' width='200'></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public ddlData: object[] = [
    { text: 'Default', value: 'Default' },
    { text: 'Both', value: 'Both' },
    { text: 'Horizontal', value: 'Horizontal' },
    { text: 'Vertical', value: 'Vertical' },
    { text: 'None', value: 'None' },
  ];
  ngOnInit(): void {
    this.data = inventoryData;
  }
  valueChange(args: ChangeEventArgs): void {
    (this.grid as GridComponent).gridLines = args.value as GridLine
  }
}

```

## 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 grid renders with **Default** mode.

See Also

- [Is it possible to apply gradient color based on min and max value in Angular Grid?](#)
- [How to customize the header cell styles using methods](#)

## Edit in angular grid component

The Grid component provides powerful options for dynamically inserting, deleting, and updating records, enabling you to modify data directly within the grid. This feature is useful when you want to enable you to perform CRUD (Create, Read, Update, Delete) operations seamlessly.

To enable editing functionality directly within the grid, you need to configure the [allowEditing](#), [allowAdding](#), and [allowDeleting](#) properties within the [editSettings](#) to **true**.

Editing feature requires a primary key column for CRUD operations. To define the primary key, set [columns.isPrimaryKey](#) to **true** in particular column.

You can start the edit action either by double clicking the particular row or by selecting the required row and click on **Edit** button in the toolbar. Similarly, you can add a new record to grid either by clicking on **Add** button in the toolbar or on an external button which is bound to invoke the [addRecord](#) method of the grid, **Save** and **Cancel** while in edit mode is possible using respective toolbar icon in grid. Deletion of the record is possible by selecting the required row and click on **Delete** button in the toolbar.

To use CRUD, inject the [EditService](#) module into the [Link to the Video](#) section.

To learn about what are all the edit modes and edit types are available in Angular Grid, you can check on this video

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
  from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-
grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [editSettings]='editSettings'
height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' [validationRules]='orderIDRules'
isPrimaryKey=true width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
[validationRules]='customerIDRules' width=120></e-column>
```

```

        <e-column field='Freight' headerText='Freight'
textAlign= 'Right' editType= 'numericedit'
        width=120 [validationRules]='freightRules' format=
'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public editSettings?: EditSettingsModel;
        public toolbar?: ToolbarItems[];
        public orderIDRules?: Object;
        public customerIDRules?: Object;
        public freightRules?: Object;
        ngOnInit(): void {
            this.data = data;
            this.orderIDRules = { required: true, number: true };
            this.customerIDRules = { required: true };
            this.freightRules = { required: true, min: 1, max: 1000 };
            this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, };
        }
    }
}

```

## MAIN.TS

```

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

```

\* If [columns.isIdentity](#) is enabled, then it will be considered as a read-only column when editing and adding a record.

\* You can disable editing for a particular column, by specifying `columns.allowEditing` to **false**.

\* You can use the **Insert** key to add a new row to the grid and use the **Delete** key to delete the selected row from the grid.

### Toolbar with edit option

The toolbar with edit option feature in the Grid component provides a [built-in toolbar](#) that includes various items for executing editing actions. This feature allows you to easily perform edit operations on the grid data, such as modifying cell values, updating changes, and canceling edits.

To enable this feature, you need to configure the [toolbar](#) property of the Grid component. This property allows you to define the items that will be displayed in the grid toolbar. By including the relevant items like **Edit**, **Add**, **Delete**, **Update**, and **Cancel** within the `toolbar` property, you can enable the edit options in the toolbar.

Here's an example of how to enable the toolbar with edit option in the Grid:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
  from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from '../datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-
grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [editSettings]='editSettings'
[toolbar]='toolbar' height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
[validationRules]='orderIDRules' textAlign='Right' isPrimaryKey='true'
width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
[validationRules]='customerIDRules' width=120></e-column>
      <e-column field='Freight' headerText='Freight'
[validationRules]='freightRules' textAlign='Right' width=120 format=
'C2'></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
[validationRules]='shipCountryRules' width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public orderIDRules?: Object;
  public customerIDRules?: Object;
  public freightRules?: Object;
  public shipCountryRules?: Object;
  ngOnInit(): void {
    this.data = data;
  }
}

```



```

        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        this.orderIDRules = { required: true, number: true };
        this.customerIDRules = { required: true };
        this.freightRules = {required: true ,number: true };
        this.shipCountryRules = { 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));

```

## Disable editing for particular column

In Grid component, you have an option to disable editing for a specific column. This feature is useful when you want to prevent editing certain columns, such as columns that contain calculated values or read-only data.

To disable editing for a particular column, you can use the [allowEditing](#) property of the **columns** object. By setting this property to **false**, you can prevent editing for that specific column.

Here's an example that demonstrates how to disable editing for the column in the Grid:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
import { GridComponent, Column } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
imports: [

        GridModule,
        DatePickerAllModule,
        FormsModule,
        TimePickerModule,
        FormsModule,
        TextBoxModule,
        MultiSelectModule,
        AutoCompleteModule,

```

```

        DropDownListModule
    ],
    providers: [EditService, ToolbarService, SortService, PageService],
    standalone: true,
    selector: 'app-root',
    templateUrl: './app.component.html',
  })
  export class AppComponent {
    public data?: Object[];
    @ViewChild('grid') public grid?: GridComponent;
    @ViewChild('dropdown') public dropdown?: DropDownListComponent;
    public editSettings?: Object;
    public toolbar?: string[];
    public orderIDRules?: Object;
    public customerIDRules?: Object;
    public freightRules?: Object;
    public editparams?: Object;
    public pageSettings?: Object;
    public alignmentData: Object[] = [
      { text: 'Order ID', value: 'OrderID' },
      { text: 'Customer ID', value: 'CustomerID' },
      { text: 'Freight', value: 'Freight' },
      { text: 'Order Date', value: 'OrderDate' },
      { text: 'Ship Country', value: 'ShipCountry' },
    ];
    public dropdownFields: Object = { text: 'text', value: 'value' }; //
    Define fields for the dropdown
    public currentColumn?: Column;
    public ngOnInit(): void {
      this.data = data;
      this.editSettings = {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
      };
      this.toolbar = ['Add', 'Delete', 'Update', 'Cancel'];
      this.orderIDRules = { required: true, number: true };
      this.customerIDRules = { required: true };
      this.freightRules = { required: true };
      this.editparams = { params: { popupHeight: '300px' } };
      this.pageSettings = { pageCount: 5 };
    }
    public changeAlignment(args: ChangeEventArgs): void {
      // Reset the allowEditing property of the previously selected column
      if (this.currentColumn) {
        this.currentColumn.allowEditing = true;
      }
      // Update the 'allowEditing' property for the selected column
      this.currentColumn = this.grid?.getColumnByField((args.value as string))
    as Column;
      this.currentColumn.allowEditing = false;
    }
  }
}

```

#### APP.COMPONENT.HTML

```

<div style="display: flex">
  <label style="padding: 30px 17px 0 0;"> Select column to disable
  editing:</label>
  <ejs-dropdownlist #dropdown style="padding: 26px 0 0 0" index="0"
  width="150"
    [dataSource]="alignmentData" [fields]="dropdownFields"
    (change)="changeAlignment($event)">
  </ejs-dropdownlist>
</div>
<div style="padding-top:20px">
  <ejs-grid #grid id='Batchgrid' [dataSource]='data' allowPaging='true'
  [pageSettings]='pageSettings'
    [editSettings]='editSettings' [toolbar]='toolbar'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' width='120'
      textAlign='Right' isPrimaryKey='true'
        [validationRules]='orderIDRules'></e-column>
      <e-column field='CustomerID' headerText='Customer ID' width='120'
      [validationRules]='customerIDRules'
        ></e-column>
      <e-column field='Freight' headerText='Freight' width='120'
      format='C2' textAlign='Right'
        editType='numericedit' [validationRules]='freightRules' >
      </e-column>
      <e-column field='OrderDate' headerText='Order Date' width='130'
      format='yMd' editType='datepickeredit'
        textAlign='Right' ></e-column>
      <e-column field='ShipCountry' headerText='Ship Country' width='150'
      editType='dropdownedit'
        [edit]='editparams' ></e-column>
    </e-columns>
  </ejs-grid>
</div>

```

## MAIN.TS

```

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

```

\* If you have set the [isPrimaryKey](#) property to **true** for a column, editing will be automatically disabled for that column.

\* You can disable the particular row using [actionBegin](#) event. please refer this [link](#).

\* You can disable the particular cell using [cellEdit](#) event. please refer this [link](#).

## Editing template column

The editing template column feature in the Grid allows you to create custom editing templates for specific columns in the grid. This feature is particularly useful when you need to customize the editing experience for certain columns, such as using custom input controls or displaying additional information during editing.

To enable the editing template column feature, you need to define the [field](#) property for the specific column in the grid's configuration. The `field` property maps the column to the corresponding field name in the data source, allowing you to edit the value of that field.

In the below demo, the **ShipCountry** column is rendered with the template.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
  from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-
  grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [editSettings]='editSettings'
[toolbar]='toolbar' height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
[validationRules]='orderIDRules' textAlign='Right' isPrimaryKey='true'
width=100></e-column>
      <e-column field='CustomerID'
[validationRules]='customerIDRules' headerText='Customer ID' width=120></e-
column>
      <e-column field='Freight' headerText='Freight'
textAlign= 'Right'
      editType= 'numericedit' width=120
[validationRules]='freightRules' format= 'C2'></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150>
        <ng-template #template let-data>
          <a href="#">{{data.ShipCountry}}</a>
        </ng-template>
      </e-column>`
})
```

```

        </e-columns>
      </ejs-grid>`
    })
    export class AppComponent implements OnInit {
      public data?: object[];
      public editSettings?: EditSettingsModel;
      public toolbar?: ToolbarItems[];
      public orderIDRules?: object;
      public customerIDRules?: object;
      public freightRules?: Object;
      ngOnInit(): void {
        this.data = data;
        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        this.orderIDRules = { required: true };
        this.freightRules = { min:1, max:1000 };
        this.customerIDRules = { 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));

```

## Customize delete confirmation dialog

Customizing the delete confirmation dialog in Grid allows you to personalize the appearance, content, and behavior of the dialog that appears when you attempt to delete an item. You can modify properties like header, showCloseIcon, and height to tailor the edit dialog to your specific requirements. Additionally, you can override default localization strings to provide custom text for buttons or other elements within the dialog.

To customize the delete confirmation dialog, you can utilize the [toolbarClick](#) event. This event is triggered when a toolbar item, such as the delete button, is clicked.

\* To enable the confirmation dialog for the delete operation in the Grid, you can set the [showDeleteConfirmDialog](#) property of the `editSettings` configuration to **true**.

\* You can refer the Grid [Default text](#) list for more localization.

The following example demonstrates how to customize the delete confirmation dialog using the `toolbarClick` event:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'

```

```

import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs, Item } from '@syncfusion/ej2-angular-navigations';
import { L10n } from '@syncfusion/ej2-base';
L10n.load({
  'en-US': {
    grid: {
      'OKButton': 'YES',
      'CancelButton': 'Discard',
      'ConfirmDelete': 'Are you sure you want to delete the selected
Record?'
    }
  }
});
@Component({
  imports: [
    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    DropDownListModule,
    DialogModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid #grid [dataSource]="data" [editSettings]="editSettings"
[toolbar]="toolbar"
    (toolbarClick)="toolbarClick($event)">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID" isPrimaryKey="true"
width="120"></e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
        <e-column field="ShipCountry" headerText="Ship Country"
width="120"></e-column>
        <e-column field="ShipCity" headerText="Ship City" width="130"></e-
column>
      </e-columns>
    </ejs-grid>
  `,
})
export class AppComponent {
  @ViewChild('grid')
  public grid?: GridComponent;

```

```

public data?: Object[];
public editSettings?: Object;
public toolbar?: Object;
public ngOnInit(): void {
    this.data = data;
    this.editSettings = {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
        showDeleteConfirmDialog: true,
    };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
}
toolbarClick(args: ClickEventArgs): void {
    if ((args.item as Item).text === 'Delete') {
        const dialogObj = ((this.grid as GridComponent).editModule as any).dialogObj;
        dialogObj.header = 'Delete Confirmation Dialog';
        dialogObj.showCloseIcon = true;
    }
}
}

```

### MAIN.TS

```

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

```

### Update boolean column value with a single click

The Syncfusion Grid allows you to update a boolean column value with a single click in the normal mode of editing. This feature streamlines the process of toggling boolean values within the grid, enhancing interaction and efficiency. This can be achieved through the use of the column template feature.

In the following sample, the **CheckBox** component is rendered as a template in the **Verified** column to make it editable with a single click.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, EditService, ToolbarService, SortService, PageService } from '@syncfusion/ej2-angular-grids';
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars';
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars';
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns';
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs';
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns';
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';

```

```

import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    CheckBoxModule,
    DropDownListModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid [dataSource]="data" [editSettings]="editSettings"
    [toolbar]="toolbar" height="315px">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID"
isPrimaryKey="true" textAlign="Right" width="120"
[validationRules]='orderIDRules'></e-column>
        <e-column field="CustomerID" headerText="Customer Name"
width="120" [validationRules]='customerIDRules'></e-column>
        <e-column field="OrderDate" headerText="Order Date"
editType="datepickeredit" format="M/d/yy" textAlign="Right"
[validationRules]='dateRules' width="130" type="date"></e-column>
        <e-column field="Freight" headerText="Freight" format="C2"
textAlign="Right" width="90" [validationRules]='freightRules'></e-column>
        <e-column field="Verified" headerText="Verified"
textAlign="Right" width="90" [validationRules]='verifiedRules'>
          <ng-template #template let-data>
            <ejs-checkbox [(checked)]="data.Verified"></ejs-checkbox>
          </ng-template>
        </e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public orderData?: object|any;
  public orderIDRules?: object;
  public customerIDRules?: object;
  public freightRules?: object;
  public dateRules?: object;
  public verifiedRules?: object;
  ngOnInit(): void {
    this.data = data;
    this.orderIDRules = { required: true };
    this.customerIDRules = { required: true };
    this.dateRules = { required: true };
  }
}

```



```

        this.verifiedRules= { required: true };
        this.freightRules = { required: true, min: 1, max: 1000 };
        this.editSettings = {
            allowEditing: true,
            allowAdding: true,
            allowDeleting: true,
        };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    }
}

```

## MAIN.TS

```

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

```

## Edit enum column

The Syncfusion Grid provides a feature that allows you to edit enum type data in a grid column. This is particularly useful when you need to edit enumerated list data efficiently.

In the following example, the `DropDownList` component is rendered within the [editTemplate](#) for the Employee Feedback column using **ngTemplate**. The enumerated list data can be bound to the Employee Feedback column using the two-way binding (`@bind-Value`).

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService,
SortService, ForeignKeyService , PageService } from '@syncfusion/ej2-angular-
grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule , ComboBoxModule } from '@syncfusion/ej2-angular-
dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems, SaveEventArgs } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,

```

```

        MultiSelectModule,
        AutoCompleteModule,
        DropDownListModule,
        ComboBoxModule
    ],
    providers: [EditService, ToolbarService, SortService,
        ForeignKeyService, PageService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]="details" [editSettings]="editSettings"
    [toolbar]="toolbar" (actionBegin)="actionBegin($event)">
        <e-columns>
            <e-column field="OrderID" headerText="Order ID" isPrimaryKey="true"
            textAlign="Right" width="90"></e-column>
            <e-column field="CustomerID" headerText="Employee Name"
            textAlign="Left" width="100"></e-column>
            <e-column field="FeedbackDetails" headerText="Employee Feedback"
            textAlign="Left" width="120">
                <ng-template #editTemplate let-data>
                    <ejs-dropdownlist [(ngModel)]="orderData.FeedbackDetails"
                    [dataSource]="dropDownEnumValue" [fields]="dropdownFields"
                    popupHeight="150px">
                        </ejs-dropdownlist>
                    </ng-template>
                </e-column>
            </e-columns>
        </ejs-grid>`
    ))
export class AppComponent implements OnInit {
    public details?: EmployeeDetails[];
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItems[];
    public orderData?: EmployeeDetails | any;
    public orderIDRules?: object;
    public customerIDRules?: object;
    public freightRules?: object;
    public dropDownEnumValue: string[] = [];
    public dropdownFields: Object = { text: 'FeedbackDetails', value:
    'FeedbackDetails' };
    ngOnInit(): void {
        this.details = data as EmployeeDetails[];
        this.orderIDRules = { required: true };
        this.freightRules = { required: true, min: 1, max: 1000 };
        this.editSettings = {
            allowEditing: true,
            allowAdding: true,
            allowDeleting: true,
        };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        this.dropDownEnumValue = Object.keys(Feedback).filter((key: string) =>
        !isNaN(Number((Feedback as any)[key])));
    }
    actionBegin(args: SaveEventArgs) {
        if (args.requestType === 'beginEdit' || args.requestType === 'add') {
            this.orderData = Object.assign({}, args.rowData);
        }
        if (args.requestType === 'save') {

```

```

        (args.data as EmployeeDetails) ['FeedbackDetails'] =
        this.orderData['FeedbackDetails'];
    }
}
}
export interface EmployeeDetails {
    OrderID: number;
    CustomerID: string;
    FeedbackDetails: Feedback;
}
export enum Feedback {
    Positive = 0,
    Negative = 1,
}
}

```

### MAIN.TS

```

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

```

### Edit complex column

The edit template for complex column in Grid is used to customize the editing experience when dealing with complex data structures. This capability is particularly useful for handling nested data objects within grid columns. By default, the grid binds complex data to column fields using the dot (.) operator. However, when you render custom elements, such as input fields, in the edit template for a complex column, you must use the (\_) underscore operator instead of the dot (.) operator to bind the complex object.

In the following sample, the input element is rendered in the edit template of the FirstName and LastName column. The edited changes can be saved using the `name` property of the input element. Since the complex data is bound to the FirstName and LastName column, The `name` property should be defined as **NameFirstName and NameLastName**, respectively, instead of using the dot notation (**Name.FirstName and Name.LastName**).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { complexData } from './datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-
grids';
@Component({

```

```

imports: [
    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    DropDownListModule
],
providers: [EditService, ToolbarService, SortService, PageService],
standalone: true,
selector: 'app-root',
template: `
    <ejs-grid #grid [dataSource]="data" [height]="315"
[editSettings]="editSettings" [toolbar]="toolbar">
        <e-columns>
            <e-column field="EmployeeID" headerText="Employee ID"
textAlign="Right" width="120"></e-column>
            <e-column field="Name.FirstName" headerText="First Name"
width="200">
                <ng-template #editTemplate let-data>
                    <input class="e-input" name="Name__FirstName" type="text"
id="Name__FirstName" [value]="data.Name.FirstName" />
                </ng-template>
            </e-column>
            <e-column field="Name.LastName" headerText="Last Name" width="200">
                <ng-template #editTemplate let-data>
                    <input class="e-input" type="text" name="Name__LastName"
id="Name__LastName" [value]="data.Name.LastName" >
                </ng-template>
            </e-column>
            <e-column field="Title" headerText="Title" width="150" ></e-column>
        </e-columns>
    </ejs-grid>
`
,
})
export class AppComponent implements OnInit {

    public data?: object[];
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItems[];
    ngOnInit(): void {
        this.data = complexData;
        this.editSettings = {
            allowEditing: true,
            allowAdding: true,
            allowDeleting: true,
        };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    }
}

```

**MAIN.TS**

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

### Edit foreign key column

The Syncfusion Grid offers a powerful editing feature for foreign key columns, enhancing the default rendering of the DropDownList component during editing. This flexibility is particularly useful when you need to customize the editor for foreign key columns. By default, the Syncfusion Grid renders the DropDownList component as the editor for foreign key columns during editing. However, you can enhance and customize this behavior by leveraging the [editTemplate](#) property for the column using **ng-template**. The **editTemplate** property allows you to specify a cell edit template that serves as an editor for a particular column, accepting either a template string or an HTML element ID.

In the following code example, the Employee Name is a foreign key column. When editing, the ComboBox component is rendered instead of DropDownList.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService,
SortService, ForeignKeyService , PageService } from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule , ComboBoxModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data, employeeData, columnDataType } from './datasource';
import {
    EditSettingsModel,
    ToolbarItems,
    SaveEventArgs,
} from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    DropDownListModule,
    ComboBoxModule

  ],
  providers: [EditService, ToolbarService, SortService,
    ForeignKeyService, PageService],
```

```

standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid [dataSource]="data" height="315"
[editSettings]="editSettings" [allowPaging]="true" [toolbar]="toolbar"
(actionBegin)=" actionBegin($event)">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID"
textAlign="Right" isPrimaryKey="true" width="120"></e-column>
        <e-column field="EmployeeID" headerText="Employee Name"
foreignKeyValue='FirstName' [dataSource]="employeeData" width="150">
          <ng-template #editTemplate let-data>
            <ejs-combobox [(value)]="orderData.EmployeeID"
[dataSource]="employees" [fields]="comboFields" >
              </ejs-combobox>
            </ng-template>
          </e-column>
        <e-column field="OrderDate" headerText="Order Date"
format="yMd" type="date" textAlign="Right" width="130"></e-column>
        <e-column field="Freight" headerText="Freight" format="C2"
textAlign="Right" width="120"></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public orderData?: object | any;
  public orderIDRules?: object;
  public customerIDRules?: object;
  public employeeIDRules?: object;
  public employees: object[] = [];
  public employeeData: Object[] = employeeData;
  public comboFields: Object = { text: 'FirstName', value: 'EmployeeID' };
  ngOnInit(): void {
    this.data = data;
    this.orderIDRules = { required: true };
    this.customerIDRules = { required: true };
    this.employeeIDRules = { required: true };
    this.editSettings = {
      allowEditing: true,
      allowAdding: true,
      allowDeleting: true,
    };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    this.employees = [
      { FirstName: 'Nancy', EmployeeID: 1 },
      { FirstName: 'Andrew', EmployeeID: 6 },
      { FirstName: 'Janet', EmployeeID: 3 },
      { FirstName: 'Margaret', EmployeeID: 4 },
      { FirstName: 'Steven', EmployeeID: 5 },
      { FirstName: 'Laura', EmployeeID: 8 }
    ];
  }
  actionBegin(args: SaveEventArgs) {
    if (args.requestType === 'beginEdit' || args.requestType === 'add') {

```

```

        this.orderData = Object.assign({}, args.rowData);
    }
    if (args.requestType === 'save') {
        (args.data as columnDataType) ['EmployeeID'] =
this.orderData ['EmployeeID'];
    }
}
}
}

```

## 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 perform CRUD action externally

Performing CRUD (Create, Read, Update, Delete) actions externally in the Syncfusion Grid allows you to manipulate grid data outside the grid itself. This can be useful in scenarios where you want to manage data operations programmatically.

#### Using separate toolbar

The Syncfusion Grid enables external CRUD operations, allowing you to efficiently manage data manipulation within the grid. This capability is particularly useful when you need to manage data operations using a separate toolbar.

To perform CRUD operations externally, the following methods are available:

[addRecord](#) - To add a new record. If no data is passed then add form will be shown.

[startEdit](#) - To edit the selected row.

[deleteRecord](#) - To delete a selected row.

[endEdit](#) - If the grid is in editable state, then you can save a record by invoking this method.

[closeEdit](#) - To cancel the edited state.

The following example demonstrates the integration of the Syncfusion Grid with a separate toolbar for external CRUD operations. The toolbar contains buttons for Add, Edit, Delete, Update, and Cancel.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';

```

```

import { data } from './datasource';
import { ClickEventArgs } from '@syncfusion/ej2-buttons';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    ToolbarModule ,
    DropDownListModule

  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <ejs-toolbar (clicked)="onToolbarClick($event)">
        <e-items>
          <e-item text="Add" id="add"></e-item>
          <e-item text="Edit" id="edit"></e-item>
          <e-item text="Delete" id="delete"></e-item>
          <e-item text="Update" id="update"></e-item>
          <e-item text="Cancel" id="cancel"></e-item>
        </e-items>
      </ejs-toolbar>
    </div>
    <div style="padding-top:20px">
      <ejs-grid #grid id='Batchgrid' [dataSource]='data'
allowPaging='true' [pageSettings]='pageSettings'
[editSettings]='editSettings'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID' width='120'
textAlign='Right' isPrimaryKey='true'
[validationRules]='orderIDRules'></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width='120' [validationRules]='customerIDRules'
></e-column>
          <e-column field='Freight' headerText='Freight' width='120'
format='C2' textAlign='Right'
editType='numericedit' [validationRules]='freightRules' >
            </e-column>
          <e-column field='OrderDate' headerText='Order Date'
width='130' format='yMd' editType='datepickeredit'
            textAlign='Right' ></e-column>
          <e-column field='ShipCountry' headerText='Ship Country'
width='150' editType='dropdownedit'
            [edit]='editparams' ></e-column>
        </e-columns>
      </ejs-grid>
    </div>`,
  })
export class AppComponent {

```



```

public data?: Object[];
@ViewChild('grid') public grid?: GridComponent;
public editSettings?: Object;
public orderIDRules?: Object;
public customerIDRules?: Object;
public freightRules?: Object;
public editparams?: Object;
public pageSettings?: Object;
public currentColumn: any;
public ngOnInit(): void {
    this.data = data;
    this.editSettings = {
        allowEditing: true,
        allowAdding: true,
        allowDeleting: true,
    };
    this.orderIDRules = { required: true, number: true };
    this.customerIDRules = { required: true };
    this.freightRules = { required: true };
    this.editparams = { params: { popupHeight: '300px' } };
    this.pageSettings = { pageCount: 5 };
}
public onToolbarClick(args: ClickEventArgs): void {
    switch ((args as any).item.id) {
        case 'add':
            (this.grid as GridComponent).addRecord();
            break;
        case 'edit':
            (this.grid as GridComponent).startEdit();
            break;
        case 'delete':
            (this.grid as GridComponent).deleteRecord();
            break;
        case 'update':
            (this.grid as GridComponent).endEdit();
            break;
        case 'cancel':
            (this.grid as GridComponent).closeEdit();
            break;
    }
}
}

```

## MAIN.TS

```

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

```

### Using external form

Performing the edit operation in a custom external form in the Syncfusion Grid is a valuable feature when you need to customize the edit operation within a separate form rather than the default in-grid editing.

To enable the use of an external form for editing in Syncfusion Grid, you can make use of the **RowSelected** property. This property specifies whether the edit operation should be triggered when a row is selected.

In the following example, it demonstrates how to edit the form using an external form by utilizing the **RowSelected** property:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule, NumericTextBoxAllModule } from '@syncfusion/ej2-
angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { GridComponent, RowSelectEventArgs } from '@syncfusion/ej2-angular-
grids';
import { data } from '../datasource'
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule,
    ButtonModule,
    NumericTextBoxAllModule,
    DropDownListModule

  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div class="row" >
      <div class="col-xs-6 col-md-3">
        <div>
          <div class="form-row">
            <div class="form-group col-md-12">
              <label for="orderedit">OrderID</label>
              <input class="form-control"
                [(ngModel)]="selectedProduct.OrderID" type="number" disabled />
            </div>
          </div>
          <div class="form-row">
            <div class="form-group col-md-12">
              <label for="customeredit">CustomerID</label>
            </div>
          </div>
        </div>
      </div>
    </div>`
})
```

```

        <ejs-textbox
[ (value) ]="selectedProduct.CustomerID"></ejs-textbox>
        </div>
    </div>
    <div class="form-row">
        <div class="form-group col-md-12">
            <label for="freightedit">Frieght</label>
            <ejs-numerictextbox
[ (value) ]="selectedProduct.Freight"></ejs-numerictextbox>
            </div>
        </div>
        <div class="form-row">
            <div class="form-group col-md-12">
                <label for="countryedit">ShipCountry</label>
                <ejs-dropdownlist
[ (value) ]="selectedProduct.ShipCountry" [dataSource]="dropdown"
[fields]="dropdownFields">
                </ejs-dropdownlist>
            </div>
        </div>
    </div>
    <button ej-button id="btn" (click)="save()">Save</button>
</div>
<div class="col-xs-6 col-md-9">
    <ejs-grid #grid [dataSource]="orders" height="315"
width="500px" (rowSelected)="rowSelectHandler($event)"
[editSettings]="editSettings">
        <e-columns>
            <e-column field="OrderID" headerText="OrderID"
isPrimaryKey="true" textAlign="Right" width="120"></e-column>
            <e-column field="CustomerID" headerText="CustomerID"
textAlign="Right" width="120"></e-column>
            <e-column field="Freight" headerText="Freight"
textAlign="Right" format="C2" width="120"></e-column>
            <e-column field="ShipCountry" headerText="ShipCountry"
textAlign="Right" width="120"></e-column>
        </e-columns>
    </ejs-grid>
</div>
</div>
,
}))
export class AppComponent {
    public orders:Object[] = data;
    @ViewChild('grid') public grid?:GridComponent;
    public dropdown: Object[] = [
        { shipCountry: 'Germany' },
        { shipCountry: 'Brazil' },
        { shipCountry: 'France' },
        { shipCountry: 'Belgium' },
        { shipCountry: 'Switzerland' },
        { shipCountry: 'Venezuela' },
        { shipCountry: 'USA' },
        { shipCountry: 'Mexico' },
        { shipCountry: 'Italy' },
        { shipCountry: 'Sweden' },
        { shipCountry: 'Finland' },
    ],

```

```

    { shipCountry: 'Spain' },
    { shipCountry: 'Canada' },
    { shipCountry: 'Portugal' },
    { shipCountry: 'Denmark' },
    { shipCountry: 'Austria' },
    { shipCountry: 'UK' },
    { shipCountry: 'Ireland' },
    { shipCountry: 'Norway' },
    { shipCountry: 'Argentina' },
  ];
  public selectedProduct: Order = new Order();
  public dropdownFields: Object = { text: 'shipCountry', value:
'shipCountry' };
  public editSettings: Object = { allowEditing: true };
  save(): void {
    const index = (this.grid as GridComponent).getSelectedRowIndexes()[0];
    (this.grid as GridComponent).updateRow(index, this.selectedProduct);
  }
  rowSelectHandler(args: RowSelectEventArgs ): void {
    (this as any).selectedProduct = { ...args.data };
  }
}
export class Order {
  OrderID?: number;
  CustomerID?: string;
  Freight?: number;
  ShipCountry?: string;
}

```

## MAIN.TS

```

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

```

### Troubleshoot editing works only for first row

The Editing functionalities can be performed based upon the primary key value of the selected row. If [isPrimaryKey](#) property is not defined in the grid, then edit or delete action take places the first row. To overcome this, ensure that you establish the [isPrimaryKey](#) property as **true** for the relevant column responsible for holding the unique identifier for each row.

### How to make a grid column always editable

To make a Grid column always editable, you can utilize the column template feature of the Grid. This feature is useful when you want to edit a particular column's values directly within the grid.

In the following example, the textbox is rendered in the **Freight** column using a column template. The keyup event for the Grid is bound using the [created](#) event of the Grid, and the edited changes are saved in the data source using the [updateRow](#) method of the Grid.

## APP.COMPONENT.TS

```

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

```

```

import { GridModule, EditService, ToolbarService, SortService, PageService }
  from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, parentsUntil } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
[editSettings]='editSettings' height='315px' (created)="created()">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=120></e-column>
      <e-column field='OrderDate' headerText='Order Date'
width=130 textAlign='Right' format='yMd'></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
width=140></e-column>
      <e-column field='Freight' headerText='Receipt Amount'
textAlign= 'Right' width=150>
        <ng-template #template let-data>
          <input id='{{data.OrderID}}'
value='{{data.Freight}}' class='custemp' type='text' style='width: 100%'>
        </ng-template>
      </e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  public editSettings?: Object;
  ngOnInit(): void {
    this.data = data;
  }
  created() {
    (this.grid as GridComponent).element.addEventListener('keyup',
function (e) { // Bind the keyup event for the grid.
```

```

        if ((e.target as HTMLElement).classList.contains('custemp')) {
// Based on this condition, you can find whether the target is an input
// element or not.
            var row = parentsUntil(e.target as HTMLElement, 'e-row');
            var rowIndex = (row as HTMLFormElement)['rowIndex']; // Get
the row index.
            var uid = row.getAttribute('data-uid');
            var grid = (document.getElementsByClassName('e-grid')[0] as
HTMLFormElement)['ej2_instances'][0];
            var rowData = grid.getRowObjectFromUID(uid).data; // Get the
row data.
            rowData.Freight = ((e.target as HTMLFormElement)['value']);
// Update the new value for the corresponding column.
            grid.updateRow(rowIndex, rowData); // Update the modified
value in the row data.
        }
    });
}
}

```

### MAIN.TS

```

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

```

\* If a template column has a corresponding **field** property defined, the value entered in the template column's input field will be stored in the associated edit column of the row's data object.

See also

- [Cascading DropDownList with Grid Editing](#)
- [Tab Inside the Dialog Editing](#)
- [Apply animation for Grid Edit dialog in Angular Grid](#)
- [CRUD operations using asp.net core web api methods in Angular Grid](#)
- [How to restrict ArrowUp increase and ArrowDown decrease value in Grid numeric cell in Angular Grid](#)
- [How to use DropDownList and Combo-Box in Batch-edit mode of Angular Grid](#)
- [How to use CellEditArgs event in Angular Grid](#)
- [How to render Grid with add form always in Angular Grid](#)
- [How to show the numeric key pad in mobile device when edit the number column in Angular Grid](#)
- [How to enable inline editing in Angular 4 Data Grid/Table](#)

### Sorting in Angular Grid component

The Grid component provides built-in support for sorting data-bound columns in ascending or descending order. To enable sorting in the grid, set the [allowSorting](#) property to **true**.

To sort a particular column in the grid, click on its column header. Each time you click the header, the order of the column will switch between **Ascending** and **Descending**.

To use the sorting feature, you need to inject the **SortService** in the provider section of **AppModule**.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowSorting]='true'
height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

### MAIN.TS

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

\* Grid column sorted in **Ascending** order. If you click on an already sorted column, then toggles the sort direction.

\* You can apply and clear sorting by using the [sortColumn](#) and [clearSorting](#) methods.

\* To disable sorting for a specific column, set the [columns.allowSorting](#) property to **false**.

### Initial sorting

By default, the grid component does not apply any sorting to its records at initial rendering. However, you can apply initial sorting by setting the [sortSettings.columns](#) property to the desired [field](#) and sort

[direction](#). This feature is helpful when you want to display your data in a specific order when the grid is first loaded.

The following example demonstrates how to set [sortSettings.columns](#) for **OrderID** and **ShipCity** columns with a specified [direction](#).

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowSorting]='true'
[sortSettings]='sortOptions' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public sortOptions?: object;
  ngOnInit(): void {
    this.data = data;
    this.sortOptions = { columns: [{ field: 'OrderID', direction:
'Ascending' }, { field: 'ShipCity', direction: 'Descending' }] };
  }
}
```

#### MAIN.TS

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

The initial sorting defined in [sortSettings.columns](#) will override any sorting applied through user interaction.



### Multi-column sorting

The Grid component allows to sort more than one column at a time using multi-column sorting. To enable multi-column sorting in the grid, set the [allowSorting](#) property to **true**, and set the [allowMultiSorting](#) property to **true** which enable the user to sort multiple columns by hold the **CTRL** key and click on the column headers. This feature is useful when you want to sort your data based on multiple criteria to analyze it in various ways.

To clear multi-column sorting for a particular column, press the "Shift + mouse left click".

- \* The [allowSorting](#) must be true while enabling multi-column sort.
- \* Set [allowMultiSorting](#) property as **false** to disable multi-column sorting.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowSorting]='true'
[allowMultiSorting]='true' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

### MAIN.TS

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

### Prevent sorting for particular column

The Grid component provides the ability to prevent sorting for a particular column. This can be useful when you have certain columns that you do not want to be included in the sorting process.

It can be achieved by setting the [allowSorting](#) property of the particular column to **false**.

The following example demonstrates, how to disable sorting for **CustomerID** column.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowSorting]='true'
height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100 [allowSorting]=false></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

#### MAIN.TS

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

### Sort order

By default, the sorting order will be as **ascending -> descending -> none**.

When you click on a column header for the first time, it sorts the column in ascending order. Clicking the same column header again will sort the column in descending order. A repetitive third click on the same column header will clear the sorting.

### Custom sorting

The Grid component provides a way to customize the default sort action for a column by defining the [column.sortComparer](#) property. The sort comparer function works similar to the [Array.sort](#) comparer function, and allows to define custom sorting logic for a specific column.

The following example demonstrates how to define custom sort comparer function for the **Customer ID** column.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowSorting]='true'
height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID'
[sortComparer]='sortComparer' headerText='Customer ID' width=100></e-column>
      <e-column field='Freight' headerText='Freight' width=80
format='C2'></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  // The custom function
  public sortComparer = (reference: string, comparer: string) => {
    if (reference < comparer) {
      return -1;
    }
    if (reference > comparer) {
      return 1;
    }
    return 0;
  }
  ngOnInit(): void {
    this.data = data;
  }
}
```



**MAIN.TS**

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

The `customSortComparer` function takes two parameters: `a` and `b`. The `a` and `b` parameters are the values to be compared. The function returns `-1`, `0`, or `1`, depending on the comparison result.

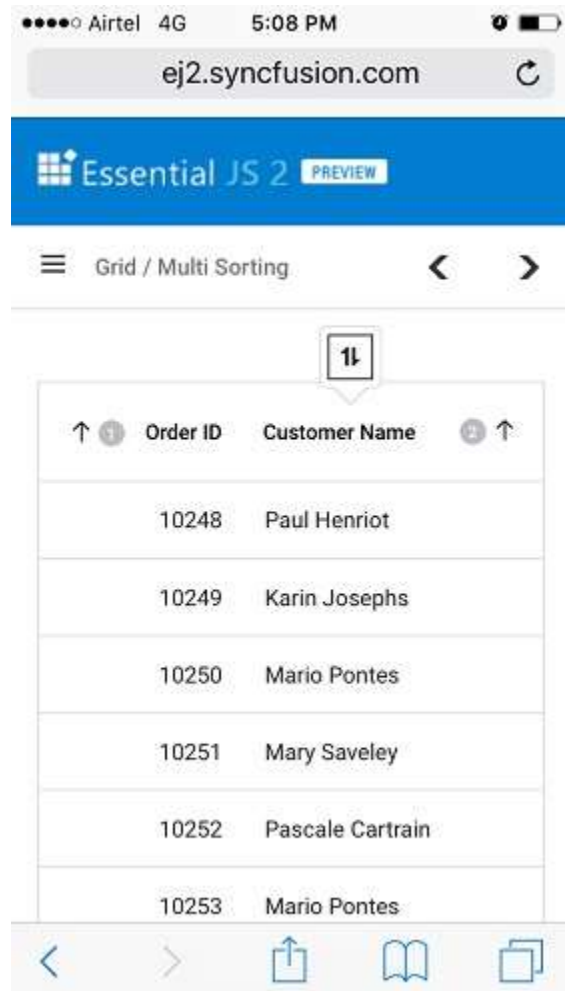
**Touch interaction**

When you tap grid header on touch screen devices, then the selected column header is sorted and

display a popup  for multi-column sorting, tap on the popup to sort multiple columns and then tap the desired grid headers. 

The `allowMultiSorting` and `allowSorting` should be **true** then only the popup will be shown.

The following screenshot represents a grid touch sorting in the device.



### Sort foreign key column based on text

To perform sorting based on foreign key column, the foreign key column can be enabled by using [column.dataSource](#), [column.foreignKeyField](#) and [column.foreignKeyValue](#) properties.

### Sort foreign key column based on text for local data

In the case of local data in the grid, the sorting operation will be performed based on the [foreignKeyValue](#) property of the column. The [foreignKeyValue](#) property should be defined in the column definition with the corresponding foreign key value for each row. The grid will sort the foreign key column based on the text representation of the [foreignKeyValue](#) property.

The following example demonstrates how to perform sorting by enabling a foreign key column, where the **CustomerID** column acts as a foreign column displaying the **ContactName** column from foreign data.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService, ForeignKeyService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data, customerData } from './datasource';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  providers: [SortService, ForeignKeyService],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid [dataSource]='data' [allowSorting]='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
foreignKeyValue='ContactName' foreignKeyField='CustomerID'
[dataSource]='customerData' width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
      </e-columns>
    </ejs-grid> `
})
export class AppComponent implements OnInit {
  public data?: object[];
  public customerData: object[] = customerData;
  ngOnInit(): void {
    this.data = data;
  }
}
```

### MAIN.TS

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

Make sure to inject the **ForeignKeyService** in the provider section of the **AppModule** to ensure its availability throughout your application.

### Sort foreign key column based on text for remote data

In the case of remote data in the grid, the sorting operation will be performed based on the [foreignKeyField](#) property of the column. The `foreignKeyField` property should be defined in the column definition with the corresponding foreign key field name for each row. The grid will send a request to the server-side with the `foreignKeyField` name, and the server-side should handle the sorting operation and return the sorted data to the grid.

`typescript

```
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { ForeignKeyService } from '@syncfusion/ej2-angular-grids';
@Component({
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [height]='315'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=100></e-column>
      <e-column field='EmployeeID' headerText='Employee Name' width=120 foreignKeyValue='FirstName'
        [dataSource]='employeeData'></e-column>
      <e-column field='Freight' headerText='Freight' textAlign='Right' width=80></e-column>
      <e-column field='ShipCity' headerText='Ship City' width=130 ></e-column>
    </e-columns>
  </ejs-grid>`,
  providers: [ForeignKeyService]
})
export class AppComponent implements OnInit {
  public data: DataManager;
  public employeeData: DataManager;
  ngOnInit(): void {
    this.data = new DataManager({
      url: '/OData/Items',
```

```

adaptor: new ODataV4Adaptor()
});
this.employeeData = new DataManager({
url: '/OData/Brands',
adaptor: new ODataV4Adaptor()
});
}
}
`

```

The following code example describes the handling of sorting operation at the server side.

```

`cs
public class ItemsController : ODataController
{
[EnableQuery]
public IQueryable<Item> Get()
{
List<Item> GridData =
JsonConvert.DeserializeObject<Item[]>(Properties.Resources.ItemsJson).AsQueryable().ToList();

List<Brand> empData =
JsonConvert.DeserializeObject<Brand[]>(Properties.Resources.BrandsJson).AsQueryable().ToList();

var queryString = HttpContext.Current.Request.QueryString;
var allUrlKeyValues = ControllerContext.Request.GetQueryNameValuePairs();
string key = allUrlKeyValues.LastOrDefault(x => x.Key == "$orderby").Value;
if (key != null)
{
if (key == "EmployeeID") {
GridData = SortFor(key); //Only for foreignKey Column ascending
}
else if(key == "EmployeeID desc") {
GridData = SortFor(key); //Only for foreignKey Column descending
}
}
var count = GridData.Count();
var data = GridData.AsQueryable();

```

```

return data;
}

public List<Item> SortFor(String Sorted)
{
    List<Item> GridData =
    JsonConvert.DeserializeObject<Item[]>(Properties.Resources.ItemsJson).AsQueryable().ToList();

    List<Brand> empData =
    JsonConvert.DeserializeObject<Brand[]>(Properties.Resources.BrandsJson).AsQueryable().ToList();

    if (Sorted == "EmployeeID") //check whether ascending or descending
    empData = empData.OrderBy(e => e.FirstName).ToList();
    else if(Sorted == "EmployeeID desc")
    empData = empData.OrderByDescending(e => e.FirstName).ToList();

    List<Item> or = new List<Item>();
    for (int i = 0; i < empData.Count(); i++) {
        //Select the Field matching records
        IEnumerable<Item> list = GridData.Where(pred => pred.EmployeeID ==
        empData[i].EmployeeID).ToList();
        or.AddRange(list);
    }
    return or;
}
}
,

```

### Perform sorting based on its culture

Perform sorting based on culture in the Grid can be achieved by utilizing the [locale](#) property. By setting the `locale` property to the desired culture code, you enable sorting based on that specific culture. This allows you to apply locale-specific sorting rules and ensure accurate ordering for different languages and regions.

In the following example, sorting is performed based on the “ar” locale using the [column.sortComparer](#) property.

#### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { L10n, loadCldr, setCulture, setCurrencyCode } from
 '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from '../datasource';

```



```

import cagregorian from './ca-gregorian.json';
import currencies from './currencies.json';
import numbers from './numbers.json';
import timeZoneNames from './timeZoneNames.json';
import numberingSystems from './numberingSystems.json'
@Component({
  imports: [

    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' locale='ar'
[allowSorting]="true" height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90 [sortComparer]="sortComparer"></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100 [sortComparer]="sortComparer"></e-column>
      <e-column field='Freight' headerText='Freight'
format='C2' textAlign='Right' width=80 [sortComparer]="sortComparer"></e-
column>
      <e-column field='OrderDate' headerText='OrderDate'
[format]='formatOptions' textAlign='Right' width=120
[sortComparer]="sortComparer"></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public formatOptions?: object;
  ngOnInit(): void {
    setCulture('ar');
    setCurrencyCode('QAR');
    loadCldr(
      cagregorian,
      currencies,
      numbers,
      timeZoneNames,
      numberingSystems
    );
    this.formatOptions = { type: 'date', format: 'yyyy/MMM/dd' };
    this.data = data;
  }
  public sortComparer = (reference: number | Date | string, comparer:
number | Date | string, sortOrder: 'Ascending' | 'Descending') => {
    const referenceDate = new Date(reference);
    const comparerDate = new Date(comparer);
    if (typeof reference === 'number' && typeof comparer === 'number') {
      // Numeric column sorting
      return sortOrder === 'Ascending' ? comparer - reference :
reference - comparer;
    } else if (!isNaN(referenceDate.getTime()) &&
!isNaN(comparerDate.getTime())) {
      // Date column sorting

```

```

        return sortOrder === 'Ascending' ? comparerDate.getTime() -
referenceDate.getTime() : referenceDate.getTime() - comparerDate.getTime();
    }
    else {
        // Default sorting for other types
        const intlCollator = new Intl.Collator(undefined, { sensitivity:
'variant', usage: 'sort' });
        const comparisonResult = intlCollator.compare(String(reference),
String(comparer));
        return sortOrder === 'Ascending' ? -comparisonResult :
comparisonResult;
    }
};
}

```

### 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 sort icon

To customize the sort icon in the Grid, you can override the default grid classes **.e-icon-ascending** and **.e-icon-descending** with custom content using CSS. Simply specify the desired icons or symbols using the **content** property as mentioned below

```

`css
.e-grid .e-icon-ascending::before {
content: '\e306';
}
.e-grid .e-icon-descending::before {
content: '\e304';
}
`

```

In the below sample, grid is rendered with a customized sort icon.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
imports: [

    GridModule
],
providers: [SortService],

```

```

standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid [dataSource]='data' [allowSorting]='true'
[sortSettings]="initialSort" height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
      </e-columns>
    </ejs-grid> `,
})
export class AppComponent implements OnInit {
  public data?: object[];
  public initialSort?: object;
  ngOnInit(): void {
    this.data = data;
    this.initialSort = {
      columns: [
        { field: 'ShipCity', direction: 'Ascending' },
        { field: 'CustomerID', direction: 'Descending' },
      ],
    };
  }
}

```

## MAIN.TS

```

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

```

## Sort columns externally

The Grid component in Syncfusion's Angular suite allows you to customize the sorting of columns and provides flexibility in sorting based on external interactions. You can sort columns, remove a sort column, and clear sorting using an external button click.

### Add sort columns

To sort a column externally, you can utilize the [sortColumn](#) method with parameters **columnName**, **direction** and **isMultiSort** provided by the Grid component. This method allows you to programmatically sort a specific column based on your requirements.

The following example demonstrates how to add sort columns to a grid. It utilizes the **DropDownList** component to select the column and sort direction. When an external button is clicked, the [sortColumn](#) method is called with the specified **columnName**, **direction**, and **isMultiSort** parameters.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, SortDirection } from '@syncfusion/ej2-angular-grids';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [

    GridModule,
    ButtonModule,
    DropDownListAllModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `
<div style="display: flex">
<label style="padding: 30px 20px 0 0"> Column name :</label>
<ejs-dropdownlist
  #dropdownColumn
  style="padding: 26px 0 0 0"
  index="0"
  width="120"
  [dataSource]="columns"
  [fields]="field"
></ejs-dropdownlist>
</div>
<div style="display: flex">
<label style="padding: 30px 17px 0 0"> Sorting direction :</label>
<ejs-dropdownlist
  #dropdownDirection
  style="padding: 26px 0 0 0"
  index="0"
  width="120"
  [dataSource]="direction"
></ejs-dropdownlist>
</div>
<button
  style="margin-top: 10px "
  ej-button
  id="button"
  cssClass="e-outline"
  (click)="addSortColumn()"
>
  Add sort column
</button>
<ejs-grid style="padding: 10px 10px" #grid [dataSource]='data'
[allowSorting]='true' [sortSettings]="sortOptions" height='315px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
    textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
    width=100></e-column>
  </e-columns>
</ejs-grid>

```

```

        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
        <e-column field="Freight" headerText="Freight" width="80"
format="C2" textAlign="Right" ></e-column>
    </e-columns>
</ejs-grid>
,
})
export class AppComponent implements OnInit {
    public data?: object[];
    public sortOptions?: any;
    @ViewChild('grid')
    public grid?: GridComponent;
    @ViewChild('dropdownColumn')
    public dropdownColumn?: DropDownListComponent;
    @ViewChild('dropdownDirection')
    public dropdownDirection?: DropDownListComponent;
    public columns: object[] = [
        { text: 'Order ID', value: 'OrderID' },
        { text: 'Customer ID', value: 'CustomerID' },
        { text: 'Freight', value: 'Freight' },
    ];
    public direction: object[] = [
        { text: 'Ascending', value: 'Ascending' },
        { text: 'Descending', value: 'Descending' },
    ];
    public field: object = { text: 'text', value: 'value' };
    ngOnInit(): void {
        this.sortOptions = {
            columns: [
                { field: 'ShipName', direction: 'Ascending' },
            ],
        };
        this.data = data;
    }
    addSortColumn() {
        (this.grid as GridComponent).sortColumn(
            (this.dropdownColumn as DropDownListComponent).value as
SortDirection,
            (this.dropdownDirection as DropDownListComponent).value as
SortDirection,
            true
        );
    }
}

```

### MAIN.TS

```

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

```

*Remove sort columns*

To remove a sort column externally, you can use the `removeSortColumn` method provided by the Grid component. This method allows you to remove the sorting applied to a specific column.

The following example demonstrates how to remove sort columns. It utilizes the **DropDownList** component to select the column. When an external button is clicked, the `removeSortColumn` method is called to remove the selected sort column.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    ButtonModule,
    DropDownListAllModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `
<div style="display: flex">
<label style="padding: 30px 20px 0 0"> Column name :</label>
<ejs-dropdownlist
  #dropdown
  style="padding: 26px 0 0 0"
  index="0"
  width="120"
  [dataSource]="columns"
  [fields]="field"
></ejs-dropdownlist>
</div>
<button
  style="margin-top: 10px "
  ej-button
  id="button"
  cssClass="e-outline"
  (click)="removeSortColumn()"
>
  Remove sort column
</button>
<ejs-grid style="padding: 10px 10px" #grid [dataSource]='data'
[allowSorting]='true' [sortSettings]="sortOptions" height='315px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
    textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
    width=100></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  columns: any[];
  field: string;
  sortOptions: any;

  ngOnInit(): void {
    this.columns = data;
    this.field = 'field';
    this.sortOptions = {
      sortColumn: 'OrderID',
      sortDirection: 'Ascending'
    };
  }

  removeSortColumn(): void {
    this.sortOptions.sortColumn = null;
  }
}
```

```

        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
    </e-columns>
</ejs-grid>
,
))
export class AppComponent implements OnInit {
    public data?: object[];
    public sortOptions?: any;
    @ViewChild('grid')
    public grid?: GridComponent;
    @ViewChild('dropdown')
    public dropDown?: DropDownListComponent;
    public columns: object[] = [
        { text: 'Customer ID', value: 'CustomerID' },
        { text: 'Order ID', value: 'OrderID' },
        { text: 'Ship Name', value: 'ShipName' },
        { text: 'Ship City', value: 'ShipCity' },
    ];
    public field: object = { text: 'text', value: 'value' };
    ngOnInit(): void {
        this.sortOptions = {
            columns: [
                { field: 'CustomerID', direction: 'Ascending' },
                { field: 'ShipName', direction: 'Descending' },
            ],
        };
        this.data = data;
    }
    removeSortColumn() {
        (this.grid as GridComponent).removeSortColumn((this.dropDown as
DropDownListComponent).value 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));

```

### Clear sorting

To clear the sorting on an external button click, you can use the [clearSorting](#) method provided by the Grid component. This method clears the sorting applied to all columns in the grid.

The following example demonstrates how to clear the sorting using `clearSorting` method in the external button click.

## APP.COMPONENT.TS

```

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

```

```

import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `
<div>
  <button ej2-button id="button" cssClass="e-outline"
(click)="onExternalSort()"> Clear Sorting </button>
  <ejs-grid style="padding: 10px 10px" #grid [dataSource]='data'
[allowSorting]='true' [sortSettings]="sortOptions" height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>
</div>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public sortOptions?: any;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.sortOptions = {
      columns: [
        { field: 'CustomerID', direction: 'Ascending' },
        { field: 'ShipName', direction: 'Descending' },
      ],
    }
  }
  onExternalSort() {
    (this.grid as GridComponent).clearSorting();
  }
}

```

**MAIN.TS**

```

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

```



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

### Sorting Events

The Grid component provides two events that are triggered during the sorting action such as [actionBegin](#) and [actionComplete](#). These events can be used to perform any custom actions before and after the sorting action is completed.

1. **actionBegin:** [actionBegin](#) event is triggered before the sorting action begins. It provides a way to perform any necessary operations before the sorting action takes place. This event provides a parameter that contains the current grid state, including the current sorting column, direction, and data.
2. **actionComplete:** [actionComplete](#) event is triggered after the sorting action is completed. It provides a way to perform any necessary operations after the sorting action has taken place. This event provides a parameter that contains the current grid state, including the sorted data and column information.

The following example demonstrates how the [actionBegin](#) and [actionComplete](#) events work when sorting is performed. The [actionBegin](#) event is used to cancel the sorting of the OrderID column. The [actionComplete](#) event is used to display a message.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SortService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { SortEventArgs } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="margin-left:100px;"><p style="color:red;" id="message">{{
message }}</p></div>
    <ejs-grid [dataSource]='data' (actionComplete)='actionComplete($event)'
(actionBegin)='actionBegin($event)' [allowSorting]='true' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
      </e-columns>
    </ejs-grid>`
```

```

    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public message?: string;
        ngOnInit(): void {
            this.data = data;
        }
        actionBegin({ requestType, columnName, cancel }: SortEventArgs) {
            if (requestType === 'sorting' && columnName === 'OrderID') {
                cancel = true;
            }
        }
        actionComplete({ requestType, columnName }: SortEventArgs) {
            this.message = requestType + ' action completed for ' + columnName +
            ' column';
        }
    }
}

```

### MAIN.TS

```

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

```

[args.requestType](#) refers to the current action being performed. For example in sorting, the [args.requestType](#) value is **sorting**.

See Also

- [How to remove the Sorting indicator from the column headers in Angular Grid](#)
- [How to change loading indicator in Angular Grid](#)

### Grouping in Angular Grid component

The grouping feature in the Syncfusion Angular Grid allows you to organize data into a hierarchical structure, making it easier to expand and collapse records. You can group the columns by simply dragging and dropping the column header to the group drop area. To enable grouping in the grid, you need to set the [allowGrouping](#) property to **true**. Additionally, you can customize the grouping options using the [groupSettings](#) property.

To use the Grouping feature, need to inject **GroupService** in the provider section of your **AppModule**.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

      GridModule

  ],

```

```

providers: [GroupService],
standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowGrouping]='true'
height='267px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
      </e-columns>
    </ejs-grid>`
  ))
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}

```

### 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 group and ungroup columns in the Grid by using the [groupColumn](#) and [ungroupColumn](#) methods respectively.

\* To disable grouping for a specific column, set the [columns.allowGrouping](#) to **false**.

### Initial group

To enable initial grouping in the Grid, you can use the [groupSettings](#) property and set the [groupSettings.columns](#) property to an array of column names(field of the column) that you want to group by. This feature is particularly useful when working with large datasets, as it allows you to quickly organize and analyze the data based on specific criteria.

The following example demonstrates how to set an initial grouping for the **CustomerID** and **ShipCity** columns during the initial rendering grid, by using the [groupSettings.columns](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

```

```

        GridModule
    ],
    providers: [GroupService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowGrouping]='true'
[groupSettings]='groupOptions' height='267px']>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public groupOptions?: GroupSettingsModel;
    ngOnInit(): void {
      this.data = data;
      this.groupOptions = { columns: ['CustomerID', 'ShipCity'] };
    }
  }
}

```

### 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 group by multiple columns by specifying an array of column names in the columns property of the `groupSettings`.

### Prevent grouping for particular column

The Grid component provides the ability to prevent grouping for a particular column. This can be useful when you have certain columns that you do not want to be included in the grouping process. It can be achieved by setting the `allowGrouping` property of the particular column to `false`. The following example demonstrates, how to disable grouping for **CustomerID** column.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

```

```

        GridModule
    ],
    providers: [GroupService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowGrouping]='true'
height='267px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' [allowGrouping]='false' width=100></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        ngOnInit(): void {
            this.data = data;
        }
    }
}

```

## MAIN.TS

```

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

```

## Hide drop area

By default, the Grid provides a drop area for grouping columns. This drop area allows you to drag and drop columns to group and ungroup them. However, in some cases, you may want to prevent ungrouping or further grouping a column after initial grouping.

To hide the drop area in the Syncfusion Angular Grid, you can set the [groupSettings.showDropArea](#) property to **false**.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to hide or show the drop area. When the switch is toggled, the [change](#) event is triggered and the `groupSettings.showDropArea` property of the grid is updated accordingly.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids';
import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
}

```

```

} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GridModule,
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
<div>
  <label style="padding: 10px 10px">
    Hide or show drop area
  </label>
  <ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-switch>
</div>
<ejs-grid #grid [dataSource]='data' [allowGrouping]='true'
[groupSettings]='groupOptions' height='315px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
    <e-column field='ShipCity' headerText='Ship City' width=100></e-column>
    <e-column field='ShipName' headerText='Ship Name' width=120></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public groupOptions?: GroupSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.groupOptions = { showDropArea: false, columns: ['CustomerID', 'ShipCity'] };
  }
  onSwitchChange(args: ChangeEventArgs) {
    if (args.checked) {
      (this.grid as GridComponent).groupSettings.showDropArea = true;
    } else {
      (this.grid as GridComponent).groupSettings.showDropArea = false;
    }
  }
}

```

```
}

```

### 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 group drop area will be shown only if there is at least one column available to group.

### Show the grouped column

The Syncfusion Angular Grid has a default behavior where the grouped column is hidden, to provide a cleaner and more focused view of your data. However, if you prefer to show the grouped column in the grid, you can achieve this by setting the [groupSettings.showGroupedColumn](#) property to **true**.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to hide or show the grouped columns. When the switch is toggled, the [change](#) event is triggered and the [groupSettings.showGroupedColumn](#) property of the grid is updated accordingly.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [
        GridModule,
        ButtonModule,
        CheckBoxModule,
        RadioButtonModule,
        SwitchModule,
    ],
    providers: [GroupService],
    standalone: true,
    selector: 'app-root',
    template: `
        <div>
            <label style="padding: 10px 10px">
                Hide or show grouped columns
            </label>
            <ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-switch>
        </div>
    `
})
export class AppComponent implements OnInit {
    @ViewChild(GridComponent) grid: GridComponent;
    groupSettings: GroupSettingsModel = {
        showGroupedColumn: false
    };
    onSwitchChange(event: ChangeEventArgs): void {
        this.groupSettings.showGroupedColumn = event.checked;
        this.grid.groupSettings = this.groupSettings;
    }
    ngOnInit(): void {
        this.grid.data = data;
        this.grid.groupSettings = this.groupSettings;
    }
}
```

```

</div>
<ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
[allowGrouping]='true' [groupSettings]='groupOptions' height='315px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=100></e-
column>
    <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
    <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
  </e-columns>
</ejs-grid>`
}))
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public groupOptions?: GroupSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.groupOptions = { showGroupedColumn: true, columns: ['CustomerID',
'ShipCity'] };
  }
  onSwitchChange(args: ChangeEventArgs) {
    if (args.checked) {
      (this.grid as GridComponent).groupSettings.showGroupedColumn = false;
    } else {
      (this.grid as GridComponent).groupSettings.showGroupedColumn = true;
    }
  }
}
}

```

## MAIN.TS

```

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

```

## Reordering on grouped columns

The Syncfusion Angular Grid allows you to easily reorder the grouped columns by dragging and dropping the grouped header cells in the group drag area. By changing the order of the grouped columns, the corresponding changes are automatically reflected in the grouping hierarchy of the grid. The grid dynamically adjusts the grouping based on the reordered columns in the group drag area. Additionally, you can also drop new columns into specific positions within the group drag area.

To enable this feature, you have to set the [groupSettings.allowReordering](#) property as **true**. This is demonstrated in the sample below.

## APP.COMPONENT.TS

```

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

```



```

import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [allowGrouping]='true'
[groupSettings]='groupSettings' height='260px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public groupSettings?: GroupSettingsModel = { columns: ['ShipCity'],
allowReordering: true };
  ngOnInit(): void {
    this.data = data;
  }
}

```

### MAIN.TS

```

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

```

### Sort grouped columns in descending order during initial grouping

By default, grouped columns are sorted in ascending order. However, you can sort them in descending order during initial grouping by setting the [field](#) and [direction](#) in the [sortSettings.columns](#) property.

The following example demonstrates how to sort the **CustomerID** column by setting the [sortSettings.columns](#) property to **Descending** during the initial grouping of the grid.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, SortService } from '@syncfusion/ej2-
angular-grids'

```

```

import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, SortSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService, SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [allowGrouping]='true'
[allowSorting]='true' [sortSettings]='sortOptions'
[groupSettings]='groupOptions' height='267px']>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public groupOptions?: GroupSettingsModel;
  public sortOptions?: SortSettingsModel;

  ngOnInit(): void {
    this.data = data;
    this.groupOptions = { columns: ['CustomerID'] };
    this.sortOptions = { columns: [{ field: 'CustomerID', direction:
'Descending' }] };
  }
}

```

## MAIN.TS

```

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

```

## Group with paging

The Grid component supports grouping columns with paging feature. When grouping is applied, the grid displays aggregated information and total items based on the current page. However, by default, the group footer and group caption footer does not consider the aggregated information and total items from other pages. To get additional details from other pages, set the [groupSettings.disablePageWiseAggregates](#) property to **false**.

If remote data is bound to grid dataSource, two requests will be sent when performing grouping action one for getting the grouped data and another for getting aggregate details and total items count.

### Group by format

By default, columns are grouped by the data or value present for the particular row. However, you can also group numeric or datetime columns based on the specified format. To enable this feature, you need to set the [enableGroupByFormat](#) property of the corresponding grid column. This feature allows you to group numeric or datetime columns based on a specific format.

The following example demonstrates how to perform a group action using the `enableGroupByFormat` property for the **OrderDate** and **Freight** columns of the grid.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowGrouping]='true'
[groupSettings]='groupOptions' height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='OrderDate' headerText='Order Date'
format='yMMM' [enableGroupByFormat]='true' width=100></e-column>
      <e-column field='Freight' headerText='Freight'
format='C2' [enableGroupByFormat]='true' width=80></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public groupOptions?: GroupSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.groupOptions = { showDropArea: false, columns: ['OrderDate',
'Freight'] };
  }
}
```

### MAIN.TS

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

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

Numeric columns can be grouped based on formats such as currency or percentage, while datetime columns can be grouped based on specific date or time formats.

### Show grouped rows based on page size

Showing grouped column rows based on the page size in Syncfusion Angular Grid is useful when you have grouped data and want to control the number of grouped rows displayed per page.

The Grid component allows you to display the number of records based on the [pageSize](#). However, by default, the `pageSize` applies to individual grid rows, not to grouped rows. If you want to show grouped column rows based on the `pageSize`, you can achieve it by using a custom implementation.

Customizing the `generateQuery` method of the **Data prototype** allows you to modify the query used for data retrieval. By doing so, you can achieve the display of grouped rows based on the page size according to your specific requirements. This can be achieved in the below example.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule
 } from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { Data, } from '@syncfusion/ej2-angular-grids';
import { Query } from '@syncfusion/ej2-data';
const oldGenerateQuery = Data.prototype.generateQuery;
Data.prototype.generateQuery = function() {
    const query = oldGenerateQuery.call(this, true);
    // Check if 'pageQuery' is available in the prototype chain
    if (Data.prototype.hasOwnProperty('pageQuery')) {
        const pageQueryFn = Data.prototype['pageQuery'] as (query: Query) =>
void;
        pageQueryFn.call(this, query);
    }
    return query;
};
@Component({
imports: [

    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule
],
providers: [FilterService, PageService, GroupService,
CheckBoxSelectionService],
standalone: true,
selector: 'app-root',
```

```

    template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[pageSettings]='initialPage'
    [allowGrouping]="true" [groupSettings]="groupOptions">
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='ShipCountry'
headerText='ShipCountry' width=140></e-column>
            <e-column field='CustomerID' headerText='Name'
width=140></e-column>
            <e-column field='ShipName' headerText='ShipName'
width=140></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public groupOptions?: object;
    public initialPage?: object;
    ngOnInit(): void {
      this.data = data;
      this.groupOptions = { showGroupedColumn: false, columns:
['ShipCountry'] };
      this.initialPage = { pageSize: 5 };
    }
  }

```

## MAIN.TS

```

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

```

### Collapse all grouped rows at initial rendering

The Syncfusion Angular Grid offers a convenient feature to expand or collapse grouped rows, allowing you to control the visibility of grouped data. The option is useful when dealing with a large dataset that contains many groups, and there is a need to provide a summarized view by initially hiding the details.

To collapse all grouped rows at the initial rendering of the Grid using the [dataBound](#) event along with the [collapseAll](#) method.

The following example demonstrates how to collapse all grouped rows at the initial rendering.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

```

```

        GridModule
    ],
    providers: [GroupService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #grid [dataSource]='data' [allowGrouping]='true'
[groupSettings]='groupOptions'
        (dataBound)='dataBound()' height='267px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=110></e-column>
            </e-columns>
        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public initial: boolean = true;
        public groupOptions?: object;
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.groupOptions = { columns: ['ShipCity'] };
        }
        dataBound() {
            if (this.initial === true) {
                (this.grid as GridComponent).groupModule.collapseAll();
                this.initial = false;
            }
        }
    }
}

```

## 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 collapse all the grouped rows at the initial rendering using the [groupCollapseAll](#) method inside the [dataBound](#) event. This is demonstrated in the below code snippet,

```

`typescript
dataBound() {
    if (this.initial === true) {
        (this.grid as GridComponent).groupCollapseAll();
    }
}

```

```
this.initial = false;
```

```
}
```

```
}
```

```
,
```

The collapse all approach is suggested for a limited number of records since collapsing every grouped record takes some time. If you have a large dataset, it is recommended to use [lazy-load grouping](#). This approach is also applicable for the [groupExpandAll](#) method.

### Group or ungroup column externally

By default, the Syncfusion Grid supports interaction-oriented column grouping, where users manually group columns by dragging and dropping them into the grouping area of the grid. Grid provides an ability to group and ungroup a column using [groupColumn](#) and [ungroupColumn](#) methods. These methods provide a programmatic approach to perform column grouping and ungrouping.

The following example demonstrates how to group and ungroup the columns in a grid. It utilizes the [DropDownList](#) component to select the column. When an external button is clicked, the [groupColumn](#) and [ungroupColumn](#) methods are called to group or ungroup the selected column.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [

    GridModule,
    ButtonModule,
    DropDownListAllModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
<div style="display: flex">
  <label style="padding: 30px 20px 0 0"> Column name :</label>
  <ejs-dropdownlist
    #dropdown
    style="padding: 26px 0 0 0"
    index="0"
    width="120"
    [dataSource]="columns"
    [fields]="field"
  ></ejs-dropdownlist>
</div>
<button
```

```

        style="margin-top: 10px "
        ejs-button
        id="button"
        cssClass="e-outline"
        (click)="groupColumn()"
    >
        Group column
    </button>
    <button
        style="margin-top: 10px "
        ejs-button
        id="button"
        cssClass="e-outline"
        (click)="unGroupColumn()"
    >
        UnGroup column
    </button>
    <ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
[allowGrouping]='true' [groupSettings]='groupOptions' height='315px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
            <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public groupOptions?: GroupSettingsModel;
        @ViewChild('grid')
        public grid?: GridComponent;
        @ViewChild('dropdown') public dropDown?: DropDownListComponent;
        ngOnInit(): void {
            this.data = data;
            this.groupOptions = { showDropArea: false, columns: ['CustomerID',
'ShipCity'] };
        }
        public columns?: object[] = [
            { text: 'CustomerID', value: 'CustomerID' },
            { text: 'OrderID', value: 'OrderID' },
            { text: 'Ship City', value: 'ShipCity' },
            { text: 'Ship Name', value: 'ShipName' },
        ];
        public field?: object = { text: 'text', value: 'value' };

        groupColumn() {
            (this.grid as GridComponent).groupColumn((this.dropDown as
DropDownListComponent).value as string);
        }

        unGroupColumn() {

```



```
(this.grid as GridComponent).ungroupColumn((this.dropDown as
DropDownListComponent).value 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));
```

### Expand or collapse externally

The Syncfusion Angular Grid offers a convenient feature to expand or collapse grouped rows, allowing you to control the visibility of grouped data. This section will provide guidance on enabling this functionality and integrating it into your application using the Grid properties and methods.

#### Expand or collapse all grouped rows

Grid provides an ability to expand or collapse grouped rows using [groupExpandAll](#) and [groupCollapseAll](#) methods respectively.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to expand or collapse grouped rows. When the switch is toggled, the [change](#) event is triggered and the [groupExpandAll](#) and [groupCollapseAll](#) methods are called to expand or collapse grouped rows.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [
        GridModule,
        ButtonModule,
        CheckBoxModule,
        RadioButtonModule,
        SwitchModule,
    ],
    providers: [GroupService],
    standalone: true,
    selector: 'app-root',
    template: `
<div>
```

```

        <label style="padding: 10px 10px">
            Expand or collapse rows
        </label>
        <ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-
switch>
    </div>
    <ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
[allowGrouping]='true' [groupSettings]='groupOptions' height='315px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
            <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
        </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
    public data?: object[];
    public groupOptions?: GroupSettingsModel;
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
        this.groupOptions = { showDropArea: false, columns: ['CustomerID',
'ShipCity'] };
    }
    onSwitchChange(args: ChangeEventArgs) {
        if (args.checked) {
            (this.grid as GridComponent).groupCollapseAll();
        } else {
            (this.grid as GridComponent).groupExpandAll();
        }
    }
}

```

## MAIN.TS

```

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

```

### *Expand or collapse selected grouped row*

Expanding or collapsing selected grouped rows in a Syncfusion Angular Grid involves implementing the functionality to expand or collapse grouped records programmatically.

To enable the expand and collapse functionality for grouped rows in a grid, you can utilize the [expandCollapseRows](#) method. This method is designed to handle two scenarios such as expanding collapsed grouped records and collapsing expanded grouped records.

To implement this functionality, follow these steps:

1. Include an `input` element to capture the grouped row index.
2. Add a `button` element with a `click` event binding to trigger the `onExpandCollapseButtonClick` method. This method retrieve the grouped rows from the grid's content table using the `querySelectorAll` method.
3. Check if there are any grouped rows available.
4. If grouped rows exist, locate the group caption element based on the entered row index.
5. Call the `expandCollapseRows` method of the grid's group module, passing the group caption element to toggle its expand/collapse state.

The following example demonstrates the function that collapses the selected row using an external button click.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { FormsModule } from '@angular/forms'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';

@Component({
  imports: [

    GridModule,
    FormsModule,
    ButtonModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
<div style="display: flex">
  <input
    type="number"
    [(ngModel)]="groupedRowIndex"
    placeholder="Enter Grouped Row Index"
  />
  <button ej2-button (click)="onExpandCollapseButtonClick()">
    Collapse or Expand Row
  </button>
</div>
<div style="padding-top: 5px">
  <p style="color: red;">{{ message }}</p>
</div>

  <ejs-grid #grid style="padding-top: 5px" [dataSource]='data'
[allowGrouping]='true' [groupSettings]='groupSettings' height='240px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90 [allowGrouping]='false'></e-column>
      <e-column field='CustomerID' headerText='Customer ID' width=100></e-
column>
    </e-columns>
  </ejs-grid>
`
})
export class AppComponent implements OnInit {
  groupedRowIndex: number;
  message: string;

  ngOnInit(): void {
    this.groupedRowIndex = 0;
  }

  onExpandCollapseButtonClick(): void {
    const grid = document.getElementById('grid');
    const groupCaption = document.querySelector(
      `div.ej2-grid-groupcaption:nth-child(${this.groupedRowIndex + 1})`
    );
    if (groupCaption) {
      grid.groupModule.expandCollapseRows(groupCaption);
    }
    this.message = 'Grouped Row Expanded/Collapsed';
  }
}
```

```

        <e-column field='ShipCity' headerText='Ship City' width=100
[allowGrouping]='false'></e-column>
        <e-column field='ShipName' headerText='Ship Name' width=120
[allowGrouping]='false'></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public groupSettings?: GroupSettingsModel;
        public groupedRowIndex?: number;
        public message?: string
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.groupSettings = { columns: ['CustomerID'] };
        }
        onExpandCollapseButtonClick() {
            const groupedRows = Array.from(
                (this.grid as GridComponent)
                    .getContentTable()
                    .querySelectorAll('.e-recordplusexpend, .e-recordpluscollapse')
            );
            if (groupedRows.length >= 0 && (this.groupedRowIndex as number) <
groupedRows.length) {
                this.message = '';
                const groupCaptionElement = groupedRows[this.groupedRowIndex as
number];
                (this.grid as
GridComponent).groupModule.expandCollapseRows(groupCaptionElement);
            } else {
                (this.message as string) =
                    'The entered index exceeds the total number of grouped rows. Please
enter a valid grouped index.';
            }
        }
    }
}

```

## MAIN.TS

```

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

```

## Clear grouping

The clear grouping feature in the Syncfusion Angular Grid allows you to removing all the grouped columns from the grid. This feature provides a convenient way to clear the grouping of columns in your application.

To clear all the grouped columns in the Grid, you can utilize the [clearGrouping](#) method of the grid.

The following example demonstrates how to clear the grouping using `clearGrouping` method in the external button click.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
    <button ej2-button id="button" cssClass="e-outline"
    (click)="onExternalGroup()"> Clear Grouping </button>
    <ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
    [allowGrouping]='true' [groupSettings]='groupOptions' height='315px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
        column>
        <e-column field='ShipName' headerText='Ship Name' width=150></e-
        column>
      </e-columns>
    </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public data?: object[];
  public groupOptions?: GroupSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.groupOptions = { columns: ['CustomerID', 'ShipCity'] };
  }
  onExternalGroup() {
    (this.grid as GridComponent).clearGrouping();
  }
}

```

**MAIN.TS**

```

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

```

## Grouping Events

The Grid component provides two events that are triggered during the group action such as [actionBegin](#) and [actionComplete](#). The [actionBegin](#) event is triggered before the group action starts, and the [actionComplete](#) event is triggered after the group action is completed. You can use these events to perform any custom action based on the grouping.

1. **actionBegin event:** [actionBegin](#) event is triggered before the group action begins. It provides a way to perform any necessary operations before the group action takes place. This event provides a parameter that contains the current grid state, including the current group field name, requestType information and etc.
2. **actionComplete event:** [actionComplete](#) event is triggered after the group action is completed. It provides a way to perform any necessary operations after the group action has taken place. This event provides a parameter that contains the current grid state, including the grouped data and column information and etc.

The following example demonstrates how the [actionBegin](#) and [actionComplete](#) events work when grouping is performed. The [actionBegin](#) event is used to cancel the grouping of the **OrderID** column. The [actionComplete](#) event is used to display a message.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, } from '@angular/core';
import { data } from './datasource';
import { GroupEventArgs, GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="margin-left:100px;"><p style="color:red;"
    id="message">{{message}}</p></div>
    <ejs-grid [dataSource]='data' [allowGrouping]='true'
    [groupSettings]='groupSettings' (actionComplete)='actionComplete($event)'
    (actionBegin)='actionBegin($event)' height='260px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
        column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
        column>
      </e-columns>
    </ejs-grid>
  `
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
```

```

        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public groupSettings?: GroupSettingsModel;
        public message?: string
        ngOnInit(): void {
            this.data = data;
        }
        actionBegin(args: GroupEventArgs) {
            if (args.requestType === 'grouping' && args.columnName ===
'OrderID') {
                args.cancel = true
                this.message = args.requestType + ' action is cancelled for ' +
args.columnName + ' column';
            }
        }
        actionComplete(args: GroupEventArgs) {
            if (args.requestType === 'grouping') {
                this.message = args.requestType + ' action completed for ' +
args.columnName + ' column';
            }
            else {
                this.message = ''
            }
        }
    }
}

```

## MAIN.TS

```

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

```

The [args.requestType](#) property represents the name of the current action being performed. For instance, during grouping, the `args.requestType` value will be **grouping**.

## Limitations

- Grouping is not compatible with the following features:
  - Autofill

## See also

- [Exporting grouped records](#)
- [How to enable lazy load grouping in Grid](#)
- [How can I do client side grouping by async pipe in Angular Grid](#)
- [How to perform initial grouping by using the async pipe in Angular Grid](#)

## Filtering in Angular Grid component

Filtering is a powerful feature in the Syncfusion Grid component that enables you to selectively view data based on specific criteria. It allows you to narrow down large datasets and focus on the information you need, thereby enhancing data analysis and decision-making.

To use filter, inject **FilterService** in the provider section of **AppModule**.

To enable filtering in the Grid, you need to set the [allowFiltering](#) property of the Grid component to true. Once filtering is enabled, you can configure various filtering options through the [filterSettings](#) property of the Grid component. This property allows you to define the behavior and appearance of the filter.

Here is an example that demonstrates the default filtering feature of the grid:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowSorting]="true"
[allowFiltering]="true" [pageSettings]="pageSettings">
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
    <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
    <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageSettings?: PageSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSize: 6 };
  }
}
```



```
}

```

### 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 apply and clear filtering, by using [filterByColumn](#) and [clearFiltering](#) methods.

\* To disable Filtering for a particular column, by specifying [columns.allowFiltering](#) to false.

### Initial filter

To apply an initial filter, you need to specify the filter criteria using the [predicate](#) object in [filterSettings.columns](#). The [predicate](#) object represents the filtering condition and contains properties such as field, operator, and value.

Here is an example of how to configure the initial filter using the [predicate](#) object:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[filterSettings]='filterOptions' height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
  </ejs-grid>`
```

```

    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public filterOptions?: FilterSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.filterOptions = {
                columns: [{ field: 'ShipCity', matchCase: false, operator:
                'startswith', predicate: 'and', value: 'reims' },
                { field: 'ShipName', matchCase: false, operator: 'startswith',
                predicate: 'and', value: 'Vins et alcools Chevalier' }]
            };
        }
    }
}

```

### MAIN.TS

```

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

```

#### *Initial filter with multiple values for same column*

In the Syncfusion Angular Grid, you can establish an initial filter containing multiple values for a particular column, which helps you to preset filter conditions for a specific column using multiple values. This functionality allows you to display a filtered records in the grid right after the grid is initially loaded.

To apply the filter with multiple values for same column at initial rendering, set the filter [predicate](#) object in [filterSettings.columns](#).

The following example demonstrates, how to perform an initial filter with multiple values for same **CustomerID** column using [filterSettings.columns](#) and [predicate](#).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-
angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [
        GridModule,
        MultiSelectModule,
        DropDownListAllModule,
        CheckBoxModule
    ],
    providers: [FilterService, PageService, CheckBoxSelectionService],
    standalone: true,

```

```

        selector: 'app-root',
        template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[filterSettings]='filterOptions' height='273px'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
            </e-columns>
        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public filterOptions?: FilterSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.filterOptions = {
                type: 'Excel',
                columns: [{
                    field: 'CustomerID',
                    matchCase: false,
                    operator: 'startswith',
                    predicate: 'or',
                    value: 'VINET',
                },
                {
                    field: 'CustomerID',
                    matchCase: false,
                    operator: 'startswith',
                    predicate: 'or',
                    value: 'HANAR',
                }
            ],
        };
    }
}

```

## MAIN.TS

```

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

```

### *Initial filter with multiple values for different columns*

By applying an initial filter with multiple values for different columns in the Syncfusion Angular Grid, you have the flexibility to set predefined filter settings for each column. This results in a filtered records of the grid right after the grid is initially loaded.

To apply the filter with multiple values for different column at initial rendering, set the filter [predicate](#) object in [filterSettings.columns](#).

The following example demonstrates how to perform an initial filter with multiple values for different **Order ID** and **Customer ID** columns using `filterSettings.columns` and `predicate`.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
  from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[filterSettings]='filterOptions' height='273px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
    <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
    <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public filterOptions?: FilterSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.filterOptions = {
      type: 'Excel',
      columns: [
        {
          field: 'CustomerID',
          matchCase: false,
          operator: 'startswith',
          predicate: 'or',
          value: 'VINET',
        },
        {
          field: 'CustomerID',
```

```

        matchCase: false,
        operator: 'startswith',
        predicate: 'or',
        value: 'HANAR',
      },
      {
        field: 'OrderID',
        matchCase: false,
        operator: 'lessThan',
        predicate: 'or',
        value: 10250,
      },
      {
        field: 'OrderID',
        matchCase: false,
        operator: 'notEqual',
        predicate: 'or',
        value: 10262,
      },
    ],
  };
}
}

```

### MAIN.TS

```

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

```

### Filter operators

The Syncfusion Grid component provides various filter operators that can be used to define filter conditions for columns. The filter operator for a column can be defined using the [operator](#) property in the [filterSettings.columns](#) object.

The available operators and its supported data types are,

Operator | Description | Supported Types

a\*b | Everything that starts with "a" and ends with "b".

a\* | Everything that starts with "a".

\*b | Everything that ends with "b".

a | Everything that has an "a" in it.

ab\* | Everything that has an "a" in it, followed by anything, followed by a "b", followed by anything.

Search			Columns	
Order ID	Shipped Date	Ship Country		
10250	7/12/1996	Brazil		
10251	7/15/1996	France		
10252	7/11/1996	Belgium		
10253	7/16/1996	Brazil		
10254	7/23/1996	Switzerland		
10255	7/15/1996	Switzerland		
10256	7/17/1996	Brazil		
10257	7/22/1996	Venezuela		
10258	7/23/1996	Austria		

#### LIKE filtering

The **LIKE** filter can process single search patterns using the "%" symbol, retrieving values matching the specified patterns. The following Grid features support LIKE filtering on string-type columns:

- Filter Menu
- Filter Bar with the [filterSettings.showFilterBarOperator](#) property enabled on the Grid [filterSettings](#).
- Custom Filter of Excel filter type.

#### For example:

Operator | Description

%ab% | Returns all the value that are contains "ab" character.

ab% | Returns all the value that are ends with "ab" character.

%ab | Returns all the value that are starts with "ab" character.

Order ID	Shipped Date	Ship Country
10250	7/12/1996	Brazil
10251	7/15/1996	France
10252	7/11/1996	Belgium
10253	7/16/1996	Brazil
10254	7/23/1996	Switzerland
10255	7/15/1996	Switzerland
10256	7/17/1996	Brazil
10257	7/22/1996	Venezuela
10258	7/23/1996	Austria

By default, the Syncfusion Angular Grid uses different filter operators for different column types. The default filter operator for string type columns is **startsWith**, for numerical type columns is **equal**, and for boolean type columns is also **equal**.

#### Diacritics filter

The diacritics filter feature in the Syncfusion Angular Grid is useful when working with text data that includes accented characters (diacritic characters). By default, the grid ignores these characters during filtering. However, if you need to consider diacritic characters in your filtering process, you can enable this feature by setting the [filterSettings.ignoreAccent](#) property to true using the [filterSettings](#).

Consider the following sample where the `ignoreAccent` property is set to true in order to include diacritic characters in the filtering process:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, FilterService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
```

```

        GridModule
    ],
    providers: [PageService,
                FilterService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[filterSettings]='filterOptions' >
        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=140></e-column>
            <e-column field='Name' headerText='Name' width=140></e-
column>
            <e-column field='ShipName' headerText='Ship Name'
width=170></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=140></e-column>
        </e-columns>
    </ejs-grid>`
    ))
    export class AppComponent implements OnInit {
        public data?: object[];
        public filterOptions?: FilterSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.filterOptions = {
                ignoreAccent: true
            };
        }
    }
}

```

## MAIN.TS

```

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

```

## Filtering with case sensitivity

The Syncfusion Angular Grid provides the flexibility to enable or disable case sensitivity during filtering. This feature is useful when you want to control whether filtering operations should consider the case of characters. It can be achieved by using the [enableCaseSensitivity](#) property within the [filterSettings](#) of the grid.

Below is an example code demonstrating how to enable or disable case sensitivity while filtering:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule, SwitchModule } from '@syncfusion/ej2-angular-buttons'
import { PageService, FilterService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';

```



```

import { data } from './datasource';
import { FilterSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonModule,
    SwitchModule,
    DropDownListAllModule
  ],
  providers: [PageService,
    FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<div class='container'>
    <label for="unchecked"> Enable Case Sensitivity </label>
    <ejs-switch id="unchecked" (change)="onToggleCaseSensitive()"></ejs-switch>
  </div>
  <ejs-grid [dataSource]='data' #grid [allowFiltering]='true'
  [filterSettings]='filterOptions'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID' width=100></e-column>
      <e-column field='ShipCountry' headerText='ShipCountry'
textAlign='Right' width=90></e-column>
      <e-column field='ShipCity' headerText='Ship City' textAlign='Right'
width=120></e-column>
      <e-column field='ShipRegion' headerText='Ship Region'
textAlign='Right' width=120></e-column>
    </e-columns>
  </ejs-grid>
  `
})
export class AppComponent implements OnInit {
  @ViewChild('grid')
  public grid?: GridComponent;
  public data?: object[];
  public isCaseSensitive: boolean = false;
  public filterOptions?: FilterSettingsModel | undefined;
  ngOnInit(): void {
    this.data = data;
    this.filterOptions = { enableCaseSensitivity: this.isCaseSensitive
  };
  }
  onToggleCaseSensitive(): void {
    this.filterOptions = { enableCaseSensitivity: !this.isCaseSensitive
  };
  }
}

```

## MAIN.TS

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

### Enable different filter for a column

The Syncfusion Angular Grid offers the flexibility to customize filtering behavior for different columns by enabling various types of filters such as **Menu**, **Excel**, **Checkbox**. This feature allows you to tailor the filtering experience to suit the specific needs of each column in your grid. For example, you might prefer a menu-based filter for a category column, an Excel-like filter for a date column, and a checkbox filter for a status column.

It can be achieved by adjusting the [column.filter.type](#) property based on your requirements.

Here's an example where the menu filter is enabled by default for all columns, but you can dynamically modify the filter types through a dropdown:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { Column, FilterSettingsModel, FilterType, GridComponent } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
import { DropDownListComponent, ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule,
    ButtonModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'app.component.html',
})
export class AppComponent implements OnInit {
  @ViewChild('grid') public grid?: GridComponent;
  @ViewChild('type') public typeDropdown?: DropDownListComponent;
  public data?: object[];
  public filterSettings?: FilterSettingsModel = { type: 'Menu' };
  public columnFilterSettings?: FilterSettingsModel;
  public fieldData: string[] | undefined;
  public typeData: string[] = [];
  public column: Column | undefined;
```

```

ngOnInit(): void {
    this.data = data;
}
dataBound() {
    this.fieldData = (this.grid as GridComponent).getColumnFieldNames();
}
onFieldChange(args: ChangeEventArgs): void {
    (this.typeDropdown as DropDownListComponent).enabled = true;
    this.typeData = ['Menu', 'CheckBox', 'Excel'];
    this.column = (this.grid as GridComponent).getColumnByField(args.value
as string);
}
onTypeChange(args: ChangeEventArgs): void {
    this.columnFilterSettings = { type: args.value as FilterType};
    (this.column as Column).filter = this.columnFilterSettings;
    (this.grid as GridComponent).refresh();
}
}

```

### APP.COMPONENT.HTML

```

<div id="content" class="container">
    <div class="input-container">
        <label for="fields" class="label">Select Column</label>
        <ejs-dropdownlist #field id="fields" [dataSource]="fieldData"
(change)="onFieldChange($event)"
placeholder="Eg: OrderID"></ejs-dropdownlist>
    </div>
    <div class="input-container">
        <label for="types" class="label">Select Filter Type</label>
        <ejs-dropdownlist #type id="types" [dataSource]="typeData"
(change)="onTypeChange($event)"
placeholder="Eg: Excel" [enabled]="false"></ejs-dropdownlist>
    </div>
</div>
<ejs-grid #grid [dataSource]='data' [allowFiltering]='true' height='273px'
allowPaging=true (dataBound)="dataBound()"
[filterSettings]="filterSettings">
    <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' width=100></e-column>
        <e-column field='OrderDate' headerText='Order Date' format='yMd'
width=100></e-column>
        <e-column field='Verified' headerText='Verified' width=100
type='boolean' displayAsCheckBox=true></e-column>
    </e-columns>
</ejs-grid>

```

### MAIN.TS

```

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

```

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

### Change default filter operator for particular column

The Syncfusion Grid component provides the flexibility to change the default filter operator for a particular column. By default, the filter operator for string-type columns is **startsWith**, for numerical-type columns is **equal**, and for boolean-type columns is also **equal**. However, you may need to customize the filter operator to better match the nature of the data in a specific column. This can be achieved using the operator property within the [filterSettings](#) configuration.

Here's an example that demonstrates how to change the default filter operator column :

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule, CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { Column, GridComponent } from '@syncfusion/ej2-angular-grids';
import { data, stringOperatorsData, numericOperatorsData } from './datasource';
import { DropDownListComponent, ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule,
    ButtonModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'app.component.html',
})
export class AppComponent implements OnInit {
  @ViewChild('grid') public grid?: GridComponent;
  @ViewChild('operator') public operatorDropdown?: DropDownListComponent;
  public data?: object[];
  public fieldData?: string[];
  public availableOperators: object[] | string | undefined;
  public column: Column | undefined;
  ngOnInit(): void {
    this.data = data;
  }
  dataBound() {
    this.fieldData = (this.grid as GridComponent).getColumnFieldNames();
  }
  onFieldChange(args: ChangeEventArgs): void {
    this.availableOperators=[];
```

```

        (this.operatorDropdown as DropDownListComponent).enabled = true;
        this.column = (this.grid as GridComponent).getColumnByField(args.value
as string);
        if (this.column) {
            this.availableOperators = this.column.type === 'string' ?
stringOperatorsData : numericOperatorsData;
        }
    }
    onOperatorChange(args: ChangeEventArgs): void {
        (this.column as Column).filter = { operator: args.value as string };
    }
}

```

### APP.COMPONENT.HTML

```

<div id='content' class='container'>
    <div class='input-container'>
        <label for='fields' class='label'>Select Column</label>
        <ejs-dropdownlist #field id='fields' [dataSource]='fieldData'
(change)='onFieldChange($event)'
placeholder='Eg: OrderID'></ejs-dropdownlist>
    </div>
    <div class='input-container'>
        <label for='operator' class='label'>Select Operator</label>
        <ejs-dropdownlist #operator id='operator'
[dataSource]='availableOperators' (change)='onOperatorChange($event)'
placeholder='Eg: Equal' [enabled]='false'></ejs-dropdownlist>
    </div>
</div>
<ejs-grid #grid [dataSource]='data' [allowFiltering]='true' height='273px'
allowPaging=true (dataBound)='dataBound()'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
        <e-column field='Freight' headerText='Freight' width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=120></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=120></e-column>
    </e-columns>
</ejs-grid>

```

### MAIN.TS

```

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

```

### Filter grid programmatically with single and multiple values using method

Programmatic filtering in the Syncfusion Angular Grid with single and multiple values allows you to apply filters to specific columns in the grid without relying on interactions through the interface.

This can be achieved by utilizing the [filterByColumn](#) method of the Grid.

The following example demonstrates, how to programmatically filter the Grid using single and multiple values for the **OrderID** and **CustomerID** columns. This is accomplished by calling the `filterByColumn` method within an external button click function.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
  from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    ButtonModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  template: `
    <button ej2-button cssClass="e-outline"
    (click)="onSingleValueFilter()">Filter with single value</button>
    <button ej2-button cssClass="e-outline" style="margin-left:5px"
    (click)="onMultipleValueFilter()">Filter with multiple values</button>
    <ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
    [allowFiltering]='true' [filterSettings]='filterOptions' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=100></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=120></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
        column>
        <e-column field='ShipName' headerText='Ship Name' width=100></e-
        column>
      </e-columns>
    </ejs-grid>`
  })
export class AppComponent implements OnInit {
  public data?: object[];
  public filterOptions?: FilterSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.filterOptions= {type:'Excel'};
  }
}
```

```

    }
    onSingleValueFilter() {
        (this.grid as GridComponent).clearFiltering();
        // filter OrderID column with single value
        (this.grid as GridComponent).filterByColumn('OrderID', 'equal',
10248);
    }
    onMultipleValueFilter() {
        (this.grid as GridComponent).clearFiltering();
        // filter CustomerID column with multiple values
        (this.grid as GridComponent).filterByColumn('CustomerID', 'equal', [
            'VINET',
            'TOMSP',
            'ERNSH',
        ]);
    }
}

```

### 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 get filtered records

Retrieving filtered records in the Syncfusion Angular Grid is essential when you want to work with data that matches the currently applied filters. You can achieve this using available methods and properties in the grid component.

#### 1. Using the `getFilteredRecords()` method

The [getFilteredRecords](#) method is used to obtain an array of records that match the currently applied filters on the grid.

This method retrieves an array of records that match the currently applied filters on the grid.

Here's an example of how to get the filtering data in a Syncfusion grid using the `getFilteredRecords` method:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component, OnInit, ViewChild, Inject } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule,

```

```

        MessageModule
    ],
    providers: [FilterService, PageService,],
    standalone: true,
    selector: 'app-root',
    templateUrl: 'app.template.html',
  })
  export class AppComponent implements OnInit {
    public data?: Object[];
    public pageOptions?: Object;
    public filteredData?: Object;
    @ViewChild('grid')
    public grid?: GridComponent;
    showRecords?: boolean;
    showWarning?: boolean;
    public ngOnInit(): void {
      this.data = data;
      this.pageOptions = { pageSize: 10, pageCount: 5 };
    }
    click(): void {
      this.filteredData = (this.grid as GridComponent).getFilteredRecords();
      if (this.filteredData) {
        this.showRecords = true;
      } else {
        this.showRecords = false;
      }
      this.showWarning = !this.showRecords;
    }
    clear(): void {
      (this.grid as GridComponent).clearFiltering();
      this.showRecords = false;
      this.showWarning = false;
    }
  }
}

```

#### APP.TEMPLATE.HTML

```

<div class="control-section">
  <div *ngIf="showWarning">
    <ejs-message id="msg_warning" content="No Records" cssClass="e-
content-center"
      severity="Warning"></ejs-message>
  </div>
  <button ej-button cssClass="e-success" (click)="click()">Get Filtered
Data</button><button ej-button
  cssClass="e-danger" (click)="clear()">Clear</button>
  <ejs-grid #grid id="grid" [dataSource]="data" allowFiltering="true"
[height]="280" allowPaging="true">
    <e-columns>
      <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="90"></e-column>
      <e-column field="CustomerID" headerText="Customer ID"
width="120"></e-column>
      <e-column field="Freight" headerText="Freight" textAlign="Right"
format="C2" width="90"></e-column>
    </e-columns>
  </ejs-grid>
</div>

```



```

        <e-column field="ShipCity" headerText="Ship City"
width="120"></e-column>
    </e-columns>
</ejs-grid>
<div *ngIf="showRecords" class="e-content">
    <h3>Filtered Records</h3>
    <ejs-grid #filtergrid [dataSource]="filteredData" allowPaging="true"
[height]="200">
        <e-columns>
            <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="90"></e-column>
            <e-column field="CustomerID" headerText="Customer ID"
width="120"></e-column>
            <e-column field="Freight" headerText="Freight"
textAlign="Right" format="C2" width="90"></e-column>
            <e-column field="ShipCity" headerText="Ship City"
width="120"></e-column>
        </e-columns>
    </ejs-grid>
</div>
</div>

```

## MAIN.TS

```

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

```

## 2.Using the properties in the FilterEventArgs object

Alternatively, you can use the properties available in the [FilterEventArgs](#) object to obtain the filter record details.

- [columns](#): This property returns the collection of filtered columns.
- [currentFilterObject](#): This property returns the object that is currently filtered.
- [currentFilteringColumn](#): This property returns the column name that is currently filtered.

To access these properties, you can use the [actionComplete](#) event handler as shown below:

```

`typescript
actionComplete(args: FilterEventArgs) {
    var column = args.columns;
    var object = args.currentFilterObject;
    var name = args.currentFilteringColumn;
}
`

```

### Clear filtering using methods

The Syncfusion Grid provides a method called [clearFiltering](#) to clear the filtering applied to the grid. This method is used to remove the filter conditions and reset the grid to its original state.

Here's an example of how to clear the filtering in a Syncfusion grid using the `clearFiltering` method:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel, GridComponent } from '@syncfusion/ej2-angular-
grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button cssClass="e-primary"
(click)="onClick()">Clear filter</button><ejs-grid #grid [dataSource]='data'
[allowPaging]='true' [allowSorting]='true'
[allowFiltering]='true' [pageSettings]='pageSettings'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
    <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
    <e-column field='ShipCountry' headerText='Ship Country'
textAlign='Right' width=120></e-column>
  </e-columns>
</ejs-grid>`,
})
export class AppComponent implements OnInit {
  @ViewChild('grid') public grid?: GridComponent;
  public data?: object[];
  public pageSettings?: PageSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSize: 6 };
  }
  public onClick(): void {
    this.grid?.clearFiltering(); //clear filtering for all columns
  }
}
```

```
}

```

## MAIN.TS

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

## Filtering events

Filtering events allow you to customize the behavior of the grid when filtering is applied. You can prevent filtering for specific columns, show messages to users, or perform other actions to suit your application's needs.

To implement filtering events in the Syncfusion Angular Grid, you can utilize the available events such as [actionBegin](#) and [actionComplete](#). These events allow you to intervene in the filtering process and customize it as needed.

In the given example, the filtering is prevented for **ShipCity** column during **actionBegin** event.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
  from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel, GridComponent, FilterEventArgs } from
  '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule,
    MessageModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<div id='message'>{{message}}</div><ejs-grid #grid
[dataSource]='data' [allowFiltering]='true' height='273px'
(actionBegin)="actionBegin($event)"
(actionComplete)="actionComplete($event)">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
```

```

        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public filterOptions?: FilterSettingsModel;
        public message: string | undefined;
        @ViewChild('grid') public gridObj?: GridComponent;
        ngOnInit(): void {
            this.data = data;
        }
        actionBegin(args: FilterEventArgs) {
            if (args.requestType == 'filtering' && args.currentFilteringColumn
== 'ShipCity') {
                args.cancel = true;
                this.message = 'The ' + args.type + ' event has been triggered
and the ' + args.requestType + ' action is cancelled for ' +
args.currentFilteringColumn;
            }
        }
        actionComplete(args: FilterEventArgs) {
            if (args.requestType == 'filtering' && args.currentFilteringColumn)
{
                this.message = 'The ' + args.type + ' event has been triggered
and the ' + args.requestType + ' action for the ' +
args.currentFilteringColumn + ' column has been successfully executed';
            } else {
                this.message = '';
            }
        }
    }
}

```

### 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

- [Customizing Filter Dialog by using an additional parameter](#)
- [Hide sorting options on Excel filter dialog](#)
- [How to apply initial filter on custom binding in Angular Grid](#)
- [How to custom the display value of checkbox filter option in Angular Grid](#)
- [How to change loading indicator in Angular Grid](#)

## Scrolling in Angular Grid component

The scrolling feature in the Angular Grid component allows you to navigate through the content that extends beyond the visible area of the grid. It provides scrollbars that are automatically displayed when the content exceeds the specified **width** or **height** of the grid element. This feature is useful when you have a large amount of data or when the content needs to be displayed within a limited space. The vertical and horizontal scrollbars will be displayed based on the following criteria:

- The vertical scrollbar appears when the total height of rows present in the grid exceeds its element height.
- The horizontal scrollbar appears when the sum of columns width exceeds the grid element width.
- The [height](#) and [width](#) are used to set the grid height and width, respectively.

The default value for **height** and **width** is **auto**.

### Set width and height

The Angular Grid component offers a straightforward method to tailor the width and height of the scroller to meet your specific requirements. This is particularly useful when you want precise control over the dimensions of the scroller. To achieve this, you can use pixel values as numbers for the [width](#) and [height](#) properties of the Grid.

In the following example, the scrollbar is enabled, and the grid's **height** is set to 315 pixels, while the **width** is set to 400 pixels:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height=315 width=400>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
        width=150></e-column>
      <e-column field='EmployeeID' headerText='Employee ID'
        textAlign='Right' width=120></e-column>
      <e-column field='ShipCity' headerText='Ship City'
        width=150></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
        width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
        width=150></e-column>
    </e-columns>`
})
```

```

        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        ngOnInit(): void {
            this.data = data;
        }
    }
}

```

## MAIN.TS

```

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

```

## Responsive with parent container

The Grid component allows you to create a responsive layout by making it fill its parent container and automatically adjust its size based on the available space and changes in the container's dimensions. This capability is particularly useful for building applications that need to adapt to various screen sizes and devices.

To achieve this, you need to specify the [width](#) and [height](#) properties of the Grid as 100%. However, keep in mind that setting the height property to 100% requires the Grid's parent element to have an explicit height defined.

In the following example, the parent container has explicit height and width set, and the Grid container's height and width are both set to 100%. This ensures that the Grid adjusts its size responsively based on the dimensions of the parent container:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
    imports: [

        GridModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
        <div style="height:500px;width:600px">
            <ejs-grid [dataSource]='data' height='100%' width='100%'>
                <e-columns>
                    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                    <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                    <e-column field='Frieght' headerText='Frieght '
textAlign='Right' width=120></e-column>

```

```

        <e-column field='ShipAddress' headerText='ShipAddress'
width=150></e-column>
        </e-columns>
    </ejs-grid>
</div>`
}))
export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
        this.data = data;
    }
}

```

### MAIN.TS

```

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

```

### Sticky header

The Angular Grid component provides a feature that allows you to make column headers remain fixed while scrolling, ensuring they stay visible at all times. To achieve this, you can utilize the [enableStickyHeader](#) property by setting it to **true**.

In the below demo, the Grid headers will be sticky while scrolling the Grid's parent div element.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule } from '@syncfusion/ej2-angular-grids';
import { SwitchModule } from '@syncfusion/ej2-angular-buttons';
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [
        GridModule,
        SwitchModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
        <div style="padding:10px 0px 20px 0px">
            <label>Enable/Disable Sticky Header</label>
            <ejs-switch id="switch" (change)="toggleStickyHeader($event)"></ejs-
switch>
        </div>
        <div style='height:350px;'>
            <ejs-grid #grid [dataSource]='data'>
                <e-columns>

```

```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width='120'></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width='150'></e-column>
        <e-column field='Freight' headerText='Freight' textAlign='Right'
width='120'></e-column>
        <e-column field='ShipAddress' headerText='Ship Address'
width='150'></e-column>
    </e-columns>
</ejs-grid>
</div>
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid') public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    toggleStickyHeader(args: ChangeEventArgs): void {
        (this.grid as GridComponent).enableStickyHeader = (args.checked as
boolean);
    }
}

```

### 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 selected row

The Angular Grid component allows you to scroll the grid content to the position of the selected row, ensuring that the selected row is automatically brought into view. This feature is particularly useful when dealing with a large dataset and wanting to maintain focus on the selected row. To achieve this, you can utilize the [rowSelected](#) event provided by the Grid.

The following example that demonstrates how to use the `rowSelected` event to scroll to the selected row:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
'@syncfusion/ej2-angular-grids'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { RowSelectEventArgs, GridComponent } from '@syncfusion/ej2-angular-
grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';

@Component({
    imports: [

```



```

        GridModule,
        DropDownListModule
    ],
    providers: [PageService, ToolbarService, EditService],
    standalone: true,
    selector: 'app-root',
    template: `
        <div style="display: flex">
            <label style="padding: 30px 20px 0 0" > Select row index
: </label>
            <ejs-dropdownlist #dropdown id='value' style="padding: 26px
0 0 0" #sample index='0'
                width='220' [dataSource]='dropDownData'
                (change)='valueChange($event)' >
            </ejs-dropdownlist>
        </div>
        <div style="padding: 20px 17px 0 0">
            <ejs-grid #grid [dataSource]='data' height='315'
width='100%' (rowSelected)='rowSelected($event)'>
                <e-columns>
                    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                    <e-column field='CustomerID' headerText='Customer
ID' width=150></e-column>
                    <e-column field='Frieght' headerText='Employee ID'
textAlign='Right' width=120></e-column>
                    <e-column field='ShipAddress' headerText='Ship
Address' width=150></e-column>
                </e-columns>
            </ejs-grid>
        </div>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        public dropDownData?: Object[] = [
            { text: 'Select count' },
            { text: '10', value: '10' },
            { text: '20', value: '20' },
            { text: '30', value: '30' },
            { text: '80', value: '80' },
            { text: '100', value: '100' },
            { text: '200', value: '200' },
            { text: '232', value: '232' },
            { text: '300', value: '300' },
            { text: '500', value: '500' },
            { text: '800', value: '800' },
            { text: '820', value: '850' },
            { text: '920', value: '920' },
            { text: '2020', value: '2020' },
            { text: '3000', value: '3000' },
            { text: '4000', value: '4000' },
            { text: '4999', value: '4999' }
        ];
        ngOnInit(): void {
    
```

```

        this.data = data;
    }
    valueChange(args: ChangeEventArgs): void {
        (this.grid as
GridComponent).selectionModule.selectRow(parseInt((args.value as string),
10));
    }
    rowSelected(args: RowSelectEventArgs) {
        const rowHeight: number = (this.grid as any).getRows()[0].scrollHeight;
        (this.grid as GridComponent).getContent().children[0].scrollTop =
rowHeight * (this.grid as GridComponent).getSelectedRowIndexes()[0];
    }
}

```

### 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 empty placeholder of scrollbar

The Syncfusion Grid component provides a feature to hide the empty placeholder of the scrollbar, offering a cleaner interface without unnecessary scrollbars. To achieve this, you can utilize the [hideScroll](#) method. This method allows you to determine whether the scrollbar should be hidden based on the content's overflow.

The following example that demonstrates how to use the `hideScroll` method inside the [dataBound](#) event:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' height='315' width='100%'
(dataBound)='dataBound()'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=120></e-column>

```

```

        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data.slice(0, 2);
        }
        dataBound(): void {
            (this.grid as any).hideScroll();
        }
    }
}

```

### MAIN.TS

```

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

```

### Searching in Angular Grid component

The Syncfusion Angular Grid includes a powerful built-in searching feature that allows users to search for specific data within the grid. This feature enables efficient filtering of grid records based on user-defined search criteria, making it easier to locate and display relevant information. Whether you have a large dataset or simply need to find specific records quickly, the search feature provides a convenient solution.

To use the searching feature, need to inject **SearchService** in the provider section of your **AppModule**. And set the [allowSearching](#) property to **true** to enable the searching feature in the grid.

To further enhance the search functionality, you can integrate a search text box directly into the grid's toolbar. This allows users to enter search criteria conveniently within the grid interface. To add the search item to the grid's toolbar, use the [toolbar](#) property and add **Search** item.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule
    ]
})

```

```

    ],
    providers: [SearchService, ToolbarService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [toolbar]='toolbarOptions'
height='272px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C' width=80></e-column>
            <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=100></e-column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    ngOnInit(): void {
      this.data = data;
      this.toolbarOptions = ['Search'];
    }
  }
}

```

## MAIN.TS

```

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

```

The clear icon is shown in the Data Grid search text box when it is focused on search text or after typing the single character in the search text box. A single click of the clear icon clears the text in the search box as well as the search results in the Grid.

### Initial search

By default, the search operation can be performed on the grid data after the grid renders. However, there might be scenarios where need to perform a search operation on the grid data during the initial rendering of the grid. In such cases, you can make use of the initial search feature provided by the grid.

To apply search at initial rendering, need to set the following properties in the [searchSettings](#) object.

#### Property | Description

startswith | Checks whether a value begins with the specified value.

endswith | Checks whether a value ends with the specified value.

contains | Checks whether a value contains with the specified value.

wildcard | Processes one or more search patterns using the “\*” symbol, returning values that match the given patterns.

like |Processes a single search pattern using the “%” symbol, retrieving values that match the specified pattern.

equal |Checks whether a value equal to the specified value.

notequal |Checks whether a value not equal to the specified value.

These operators provide flexibility in defining the search behavior and allow you to perform different types of comparisons based on your requirements.

The following example demonstrates how to set the `searchSettings.operator` property based on changing the dropdown value using the `change` event of the `DropDownList` component.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems, SearchSettingsModel } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `
<div style="display: flex">
<label style="padding: 10px 10px 26px 0">
  Change the search operators:
</label>
<ejs-dropdownlist
  style="margin-top:5px"
  id="value"
  #dropdown
  index="0"
  width="100"
  [dataSource]="ddlData"
  [fields]='fields'
  (change)="valueChange($event)"
></ejs-dropdownlist>
</div>
<ejs-grid #grid style="padding: 5px 5px" [dataSource]='data'
[toolbar]='toolbarOptions' [searchSettings]="searchSettings" height='272px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID' width=100></e-
column>
```

```

        <e-column field='ShipName' headerText='Ship Name' width=110></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country'
textAlign='Right' width=100></e-column>
    </e-columns>
</ejs-grid>`
}))
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    public searchSettings?: SearchSettingsModel
    @ViewChild('grid') public grid?: GridComponent;
    public fields?: object = { text: 'text', value: 'value' };
    public ddlData?: object[] = [
        { text: 'startswith', value: 'startswith' },
        { text: 'endswith', value: 'endswith' },
        { text: 'wildcard', value: 'wildcard' },
        { text: 'like', value: 'like' },
        { text: 'equal', value: 'equal' },
        { text: 'not equal', value: 'notequal' },
    ];
    ngOnInit(): void {
        this.data = data;
        this.toolbarOptions = ['Search'];
        this.searchSettings = { operator: 'contains' };
    }
    valueChange(args: ChangeEventArgs): void {
        (this.grid as GridComponent).searchSettings.operator = args.value 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));

```

### Search by external button

The Syncfusion Grid component allows you to perform searches programmatically, enabling you to search for records using an external button instead of relying solely on the built-in search bar. This feature provides flexibility and allows for custom search implementations within your application. To search for records using an external button, you can utilize the [search](#) method provided by the Grid component.

The `search` method allows you to perform a search operation based on a search key or criteria. The following example demonstrates how to implement `search` by an external button using the following steps:

1. Add a button element outside of the grid component.
2. Attach a click event handler to the button.
3. Inside the event handler, get the reference of the grid component.

4. Invoke the `search` method of the grid by passing the search key as a parameter.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-
angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs'
@Component({
  imports: [

    GridModule,
    ButtonModule,
    TextBoxModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div class="e-float-input" style="width: 120px; display: inline-block;">
      <ejs-textbox #searchInput width="100" placeholder="Search
text"></ejs-textbox>
      <span class="e-float-line"></span>
    </div>
    <button ejs-button id='search' (click)='search()'>Search</button>
    <ejs-grid #grid [dataSource]='data' height='260px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public grid?: GridComponent;
  @ViewChild('searchInput') public searchInput?: TextBoxComponent;
  ngOnInit(): void {
    this.data = data;
  }
  search() {
    const searchText: string = (this.searchInput as
TextBoxComponent).value;
    (this.grid as GridComponent).search(searchText);
  }
}
```

```
}

```

### MAIN.TS

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

### Search specific columns

By default, the [search](#) functionality searches all visible columns. However, if you want to search only specific columns, you can define the specific column's field names in the [searchSettings.fields](#) property. This allows you to narrow down the search to a targeted set of columns, which is particularly useful when dealing with large datasets or grids with numerous columns.

The following example demonstrates how to search specific columns such as **CustomerID**, **Freight**, and **ShipCity** by using the [searchSettings.fields](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, SearchSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [height]='260'
[searchSettings]='searchSettings' [toolbar]='toolbar' >
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Center' format='C2' width=80></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100 ></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public searchSettings?: SearchSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
```



```

        this.data = data;
        this.toolbar = ['Search'];
        this.searchSettings = {fields: ['CustomerID', 'Freight',
'ShipCity']};
    }
}

```

### MAIN.TS

```

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

```

### Search on each key stroke

The search on each keystroke feature in Syncfusion Grid enables you to perform real-time searching of grid data as they type in the search text box. This functionality provides a seamless and interactive searching experience, allowing you to see the search results dynamically updating in real time as they enter each keystroke in the search box

To achieve this, you need to bind the `keyup` event to the search input element inside the `created` event of the grid component.

In the following example, the `created` event is bound to the grid component, and inside the event handler, the `keyup` event is bound to the `search` input element. Whenever the `keyup` event is triggered, the current `search` string is obtained from the `search` input element, and the `search` method is invoked on the grid instance with the current search string as a parameter. This allows the search results to be displayed in real-time as you type in the search box.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
[toolbar]='toolbarOptions' (created)='created()' height='400' width='100%'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>

```

```

        <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=80></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['Search'];
        }
        created(): void {
            (document.getElementById((this.grid as GridComponent).element.id +
            "_searchbar") as Element).addEventListener('keyup', () => {
                (this.grid as GridComponent).search(((event as
            MouseEvent).target as HTMLInputElement).value)
            });
        }
    }
}

```

## MAIN.TS

```

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

```

Search on each key stroke approach may affect the performance of the application when dealing with a large number of records.

### Perform search based on column formatting

By default, the search operation considers the underlying raw data of each cell for searching. However, in some cases, you may want to search based on the formatted data visible to the users. To search data based on column formatting, you can utilize the `grid.valueFormatterService.fromView` method within the [actionBegin](#) event. This method allows you to retrieve the formatted value of a cell and perform searching on each column using the **OR** predicate.

The following example demonstrates how to implement searching based on column formatting in the Grid. In the `actionBegin` event, retrieve the search value from the [getColumns](#) method. Iterate through the columns and check whether the column has a format specified. If the column has a format specified, use the `grid.valueFormatterService.fromView` method to get the formatted value of the cell. If the formatted value matches the search value, set the **OR** predicate that includes the current column filter and the new filter based on the formatted value.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems, SearchEventArgs, KeyboardEventArgs }
from '@syncfusion/ej2-angular-grids';
import { Query, Predicate } from '@syncfusion/ej2-data';
@Component({
  imports: [

    GridModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
[toolbar]='toolbarOptions' height='272px'
(actionBegin)="actionBegin($event)" (keyPressed)="keyPressed($event)">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C' width=80></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['Search'];
  }
  actionBegin(args: SearchEventArgs) {
    if (args.requestType == 'searching') {
      args.cancel = true;
      setTimeout(() => {
        var columns = (this.grid as GridComponent).getColumns();
        var predicate = null;
        for (var i = 0; i < columns.length; i++) {
          var val = (this.grid as
GridComponent).valueFormatterService.fromView(
            args.searchString as string,
            columns[i].getParser(),
            columns[i].type
          );
          if (val) {
            if (predicate == null) {

```

```

        predicate = new Predicate(
            columns[i].field,
            'contains',
            val,
            true,
            true
        );
    } else {
        predicate = predicate.or(
            columns[i].field,
            'contains',
            val,
            true,
            true
        );
    }
}
}
(this.grid as GridComponent).query = new
Query().where(predicate as Predicate);
}, 200);
}
}
keyPressed(args: KeyboardEventArgs) {
    if (
        args.key == 'Enter' &&
        args.target instanceof HTMLElement &&
        args.target.closest('.e-search') &&
        (args.target as HTMLInputElement).value == ''
    ) {
        args.cancel = true;
        (this.grid as GridComponent).query = new Query();
    }
}
}

```

### MAIN.TS

```

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

```

### Perform search operation in Grid using multiple keywords

In addition to searching with a single keyword, the Grid component offers the capability to perform a search operation using multiple keywords. This feature enables you to narrow down your search results by simultaneously matching multiple keywords. It can be particularly useful when you need to find records that meet multiple search conditions simultaneously. This can be achieved by the [actionBegin](#) event of the Grid.

The following example demonstrates, how to perform a search with multiple keywords in the grid by using the `query` property when the `requestType` is searching in the `actionBegin` event. The `searchString` is divided into multiple keywords using a comma (,) as the delimiter. Each keyword is then utilized to create a `predicate` that checks for a match in the desired columns. If multiple keywords are

present, the predicates are combined using an **OR** condition. Finally, the Grid's **query** property is updated with the constructed **predicate**, and the Grid is refreshed to update the changes in the UI.

On the other hand, the [actionComplete](#) event is used to manage the completion of the search operation. It ensures that the search input value is updated if necessary and clears the **query** when the search input is empty.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, SearchSettingsModel, GridComponent, SearchEventArgs, Column } from '@syncfusion/ej2-angular-grids';
import { Predicate, Query, } from '@syncfusion/ej2-data';
@Component({
  imports: [

    GridModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #Grid [dataSource]='data'
[toolbar]='toolbarOptions'
[searchSettings]='searchOptions' (actionBegin)="actionBegin($event)"
(actionComplete)="actionComplete($event)" height='400' width='100%'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=80></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public values?: string;
  public key = '';
  public removeQuery = false;
  public valueAssign = false;
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  public searchOptions?: SearchSettingsModel;
  @ViewChild('Grid') public grid?: GridComponent;
  ngOnInit(): void {
```

```

        this.data = data;
        this.searchOptions = {
            fields: [
                'OrderID',
                'CustomerID',
                'EmployeeID',
                'ShipCity',
                'ShipCountry',
                'ShipName'
            ],
            operator: 'contains',
            key: '',
            ignoreCase: true,
        };
        this.toolbarOptions = ['Search'];
    }
    actionBegin({ requestType, searchString }: SearchEventArgs) {
        if (requestType === 'searching') {
            const keys = (searchString as string).split(',');
            var flag = true;
            var predicate: any;
            if (keys.length > 1) {
                if ((this.grid as GridComponent).searchSettings.key !== '')
                {
                    this.values = searchString;
                    keys.forEach((key: string) => {
                        (this.grid as
GridComponent).getColumns().forEach((col: Column) => {
                            if (flag) {
                                predicate = new Predicate(col.field,
'contains', key, true);
                                flag = false;
                            }
                            else {
                                var predic = new Predicate(col.field,
'contains', key, true);
                                predicate = predicate.or(predic);
                            }
                        });
                    });

                    (this.grid as GridComponent).query = new
Query().where(predicate);
                    (this.grid as GridComponent).searchSettings.key = '';
                    this.valueAssign = true;
                    this.removeQuery = true;
                    (this.grid as GridComponent).refresh();
                }
            }
        }
    }
    actionComplete(args: SearchEventArgs) {
        if (args.requestType === 'refresh' && this.valueAssign) {
            const searchBar = document.querySelector<HTMLInputElement>('#' +
(this.grid as GridComponent).element.id + '_searchbar');
            if (searchBar) {
                searchBar.value = this.values || '';
            }
        }
    }
}

```

```

        this.valueAssign = false;
    }
    else if (
        args.requestType === 'refresh' &&
        this.removeQuery
    ) {
        const searchBar =
document.querySelector<HTMLInputElement>('#' + (this.grid as
GridComponent).element.id + '_searchbar');
        if (searchBar) {
            searchBar.value = '';
        }
        (this.grid as GridComponent).query = new Query();
        this.removeQuery = false;
        (this.grid as GridComponent).refresh();
    }
}
}
}

```

### MAIN.TS

```

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

```

By using this approach, you can perform a search operation in the grid using multiple keywords.

#### How to ignore accent while searching

By default, the searching operation in the Grid component does not ignore diacritic characters or accents. However, there are cases where ignoring diacritic characters becomes necessary. This feature enhances the search experience by enabling data searching without considering accents, ensuring a more comprehensive and accurate search and it can be achieved by utilizing the [searchSettings.ignoreAccent](#) property of the Grid component as **true**.

The following example demonstrates how to define the `ignoreAccent` property within the [searchSettings](#) property of the grid. Additionally, the [EJ2 Toggle Switch Button](#) component is included to modify the value of the `searchSettings.ignoreAccent` property. When the switch is toggled, the [change](#) event is triggered, and the `searchSettings.ignoreAccent` property is updated accordingly. This functionality helps to visualize the impact of the `searchSettings.ignoreAccent` setting when performing search operations.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,

```

```

    } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, GridComponent } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    GridModule,
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,

  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `
<div>
<label style="padding: 10px 10px">
Enable or disable ignoreAccent property
</label>
<ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-switch>
</div>
<ejs-grid #grid [dataSource]='data' [toolbar]='toolbarOptions'
height='272px'>
  <e-columns>
    <e-column field='CategoryName' headerText='Category Name'
width='100'></e-column>
    <e-column field='ProductName' headerText='Product Name'
width='130'></e-column>
    <e-column field='QuantityPerUnit' headerText='Quantity per unit'
width='150' textAlign='Right'></e-column>
    <e-column field='UnitsInStock' headerText='Units In Stock'
width='80' textAlign='Right'></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['Search'];
  }
  onSwitchChange(args: ChangeEventArgs) {
    if (args.checked) {
      (this.grid as GridComponent).searchSettings.ignoreAccent = true;
    } else {
      (this.grid as GridComponent).searchSettings.ignoreAccent =
false;
    }
  }
}

```



**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 set [searchSettings.ignoreAccent](#) property along with other search settings such as [fields](#), [operator](#), and [ignoreCase](#) to achieve the desired search behavior.

\* This feature works only for the characters that are not in the ASCII range.

\* This feature may have a slight impact on search performance.

**Highlight the search text**

The Syncfusion Grid component allows you to visually highlight search results within the displayed data. This feature helps you to quickly identify where the search items are found within the displayed data. By adding a style to the matched text, you can quickly identify where the search items are present in the grid.

To achieve search text highlighting in the Grid, you can utilize the [queryCellInfo](#) event. This event is triggered for each cell during the Grid rendering process, allowing you to customize the cell content based on your requirements.

The following example demonstrates how to highlight search text in grid using the [queryCellInfo](#) event. The [queryCellInfo](#) event checks if the current cell is in the desired search column, retrieves the cell value, search keyword and uses the [includes](#) method to check if the cell value contains the search keyword. If it does, the matched text is replaced with the same text wrapped in a [span](#) tag with a [customcss](#) class. You can then use CSS to define the [customcss](#) class and style to easily identify where the search keywords are present in the grid.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, SearchEventArgs, GridComponent,
QueryCellInfoEventArgs, Column } from '@syncfusion/ej2-angular-grids';
interface ColumnData{
  [key: string]: number | string;
  OrderID:number,
  Freight:number,
  CustomerID:string,
  ShipCity:string,
  ShipName:string,
  ShipCountry:string,
  ShipPostalCode:number
}
@Component({
  imports: [
```

```

        GridModule
    ],
    providers: [SearchService, ToolbarService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #grid [dataSource]='data'
[toolbar]='toolbarOptions' (actionBegin)="actionBegin($event)"
(queryCellInfo)="queryCellInfo($event)" height='400' width='100%'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=80></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            <e-column field='ShipCountry' headerText='Ship Country'
width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
        </e-columns>
    </ejs-grid>`
    ))
export class AppComponent implements OnInit {
    public key?:string = '';
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    @ViewChild('grid') public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
        this.toolbarOptions = ['Search'];
    }
    actionBegin(args:SearchEventArgs) {
        if (args.requestType === 'searching') {
            (this.key as string) = (args.searchString as
string).toLowerCase();
        }
    }
    queryCellInfo(args: QueryCellInfoEventArgs) {
        if ((this.key as string) !== '') {
            var cellContent = (args.data as ColumnData)[(args.column as
Column).field];
            var parsedContent = cellContent.toString().toLowerCase();
            if (parsedContent.includes((this.key as string).toLowerCase())) {
                var i = 0;
                var searchStr = '';
                while (i < (this.key as string).length) {
                    var index = parsedContent.indexOf((this.key as string)[i]);
                    searchStr = searchStr + cellContent.toString()[index];
                    i++;
                }
                (args.cell as HTMLElement).innerHTML = (args.cell as
HTMLElement).innerText.replaceAll(
                    searchStr,
                    "<span class='customcss'>" + searchStr + '</span>'

```

```

    );
  }
}
}
}

```

### MAIN.TS

```

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

```

### Clear search by external button

The Syncfusion Grid component provides a capability to clear searched data in the grid. This functionality offers the ability to reset or clear any active search filters that have been applied to the grid's data.

To clear the searched grid records from an external button, you can set the [searchSettings.key](#) property to an **empty** string to clear the search text. This property represents the current search text in the search box.

The following example demonstrates how to clear the searched records using an external button.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, SearchService, ToolbarService } from '@syncfusion/ej2-angular-grids';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, SearchSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  providers: [SearchService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template:
    `<button ej2-button id='clear' (click)='clearSearch()'>Clear Search</button>
      <ejs-grid #grid style="margin-top:5px" [dataSource]='data' [searchSettings]='searchOptions' [toolbar]='toolbarOptions' height='260px'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID' textAlign='Right' width=90></e-column>
          <e-column field='CustomerID' headerText='Customer ID' width=100></e-column>
        </e-columns>
      </ejs-grid>`
})
export class AppComponent implements OnInit {
  clearSearch() {
    this.grid.searchSettings.key = '';
  }
}

```

```

        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
</ejs-grid>`
}))
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    public searchOptions?: SearchSettingsModel;
    @ViewChild('grid') public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
        this.searchOptions = { fields: ['CustomerID'], operator: 'contains',
key: 'Ha', ignoreCase: true };
        this.toolbarOptions = ['Search'];
    }
    clearSearch() {
        (this.grid as GridComponent).searchSettings.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 can also clear the searched records by using the clear icon within the search input field.

See also

- [How to perform search by using Wildcard and LIKE operator filter](#)

### Paging in Angular Grid component

Paging provides an option to display grid data in segmented pages, making it easier to navigate through large datasets. This feature is particularly useful when dealing with extensive data sets.

To enable paging, you need to set the [allowPaging](#) property to **true**. This property determines whether paging is enabled or disabled for the grid. When paging is enabled, a pager component rendered at the bottom of the grid, allowing you to navigate through different pages of data.

To use paging, you need to inject the **PageService** into the provider section of your **AppModule**. This service provides the necessary methods and events to handle paging functionality.

Paging options can be configured through the [pageSettings](#) property. The `pageSettings` object allows you to control various aspects of paging, such as the page size, current page, and total number of records.

You can achieve better performance by using grid paging to fetch only a pre-defined number of records from the data source.

### Customize the pager options

Customizing the pager options in the Syncfusion Grid allows you to tailor the pagination control according to your specific requirements. You can customize the pager to display the number of pages using the `pageCount` property, change the current page using `currentPage` property, display the number of records in the grid using the `pageSize` property, and even adjust the page sizes in a dropdown using the `pageSizes` property. Additionally, you can include the current page as a query string in the URL for convenient navigation.

### Change the page size

The Syncfusion Grid allows you to control the number of records displayed per page, providing you with flexibility in managing your data. This feature is particularly useful when you want to adjust the amount of data visible to you at any given time. To achieve this, you can utilize the `pageSettings.pageSize` property. This property is used to specify the initial number of records to display on each page. The default value of `pageSize` property is **12**.

The following example demonstrates how to change the page size of a Grid using an external button click based on **TextBox** input.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { orderDetails } from './datasource';
import { GridComponent, PageSettingsModel } from '@syncfusion/ej2-angular-
grids';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GridModule,
    TextBoxModule,
    ButtonAllModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <label style="padding: 30px 17px 0 0">Enter page size:</label>
      <ejs-textbox #textbox width="120"></ejs-textbox>
      <button ejs-button #button id="button"
        (created)=clickHandler($event)>click button</button>
    </div>
    <div style="padding:20px 0 0 0">
      <ejs-grid #grid id="PagingGrid" [dataSource]="data"
        [allowPaging]="true" height="325">
        <e-columns>
          <e-column field="OrderID" headerText="Order ID"
            textAlign="Right" width="120">

```

```

        </e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
        <e-column field="ShipCity" headerText="Ship City"
width="150"></e-column>
        <e-column field="ShipName" headerText="Ship Name"
width="150"></e-column>
    </e-columns>
</ejs-grid>
</div> `
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        public pageOptions?: PageSettingsModel;
        @ViewChild('textbox') public textbox?: TextBoxComponent;
        @ViewChild('button') public button?: ButtonComponent;
        ngOnInit(): void {
            this.data = orderDetails;
        }
        clickHandler(args:any): void {
            (this.button as ButtonComponent).element.addEventListener('click', (e:
MouseEvent) => {
                e.preventDefault(); // Prevent any default behavior of the button
click
                (this.grid as GridComponent).pageSettings.pageSize =
parseInt((this.textbox as TextBoxComponent).value, 10);
            });
        }
    }
}

```

## 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 page count

The Syncfusion Grid allows you to adjust the number of pages displayed in the pager container. This is useful when you want to manage the number of pages you see while navigating through extensive datasets. The default value of `pageCount` property is **8**.

To change the page count in the Syncfusion Grid, you can utilize the [pageSettings.pageSize](#) property, which defines the number of pages displayed in the pager container.

The following example demonstrates how to change the page count of a Grid using an external button click based on **TextBox** input.

## APP.COMPONENT.TS

```

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

```

```

import { GridModule, PageService, ToolbarService, EditService } from
'@syncfusion/ej2-angular-grids'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { orderDetails } from './datasource';
import { GridComponent, PageSettingsModel } from '@syncfusion/ej2-angular-
grids';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    GridModule,
    TextBoxModule,
    ButtonModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <label style="padding: 30px 17px 0 0">Enter page count:</label>
      <ejs-textbox #textbox width="120"></ejs-textbox>
      <button ejs-button #button id="button"
(created)=clickHandler($event)>click button</button>
    </div>
    <div style="padding:20px 0 0 0">
      <ejs-grid #grid id="PagingGrid" [dataSource]="data"
[allowPaging]="true" height="325">
        <e-columns>
          <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="120">
        </e-column>
          <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
          <e-column field="ShipCity" headerText="Ship City"
width="150"></e-column>
          <e-column field="ShipName" headerText="Ship Name"
width="150"></e-column>
        </e-columns>
      </ejs-grid>
    </div> `
  })
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public pageOptions?: PageSettingsModel;
  @ViewChild('textbox') public textbox?: TextBoxComponent;
  @ViewChild('button') public button?: ButtonComponent;
  ngOnInit(): void {
    this.data = orderDetails;
  }
  clickHandler(args:any): void {
    (this.button as ButtonComponent).element.addEventListener('click', (e:
MouseEvent) => {

```

```

        e.preventDefault(); // Prevent any default behavior of the button
        click
        (this.grid as GridComponent).pageSettings.pageCount =
        parseInt((this.textbox as TextBoxComponent).value, 10);
    });
}
}

```

## 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 current page

The Syncfusion Grid allows you to change the currently displayed page, which can be particularly useful when you need to navigate through different pages of data either upon the initial rendering of the grid or update the displayed page based on interactions or specific conditions. The default value of `currentPage` property is **1**.

To change the current page in the Syncfusion Grid, you can utilize the [pageSettings.currentPage](#) property, which defines the current page number of the pager.

The following example demonstrates how to dynamically change the current page using an external button click based on **TextBox** input:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
'@syncfusion/ej2-angular-grids'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { orderDetails } from './datasource';
import { GridComponent, PageSettingsModel } from '@syncfusion/ej2-angular-
grids';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [
    GridModule,
    TextBoxModule,
    ButtonModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <label style="padding: 30px 17px 0 0">Enter current page:</label>
      <ejs-textbox #textbox width="120"></ejs-textbox>
    </div>
  `
})

```



```

        <button ej-button #button id="button"
        (created)=clickHandler($event)>click button</button>
    </div>
    <div style="padding:20px 0 0 0">
        <ejs-grid #grid id="PagingGrid" [dataSource]="data"
        [allowPaging]="true" height="325">
            <e-columns>
                <e-column field="OrderID" headerText="Order ID"
                textAlign="Right" width="120">
            </e-column>
                <e-column field="CustomerID" headerText="Customer ID"
                width="150"></e-column>
                <e-column field="ShipCity" headerText="Ship City"
                width="150"></e-column>
                <e-column field="ShipName" headerText="Ship Name"
                width="150"></e-column>
            </e-columns>
        </ejs-grid>
    </div> `
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        public pageOptions?: PageSettingsModel;
        @ViewChild('textbox') public textbox?: TextBoxComponent;
        @ViewChild('button') public button?: ButtonComponent;
        ngOnInit(): void {
            this.data = orderDetails;
        }
        clickHandler(args:any): void {
            (this.button as ButtonComponent).element.addEventListener('click', (e:
            MouseEvent) => {
                e.preventDefault(); // Prevent any default behavior of the button
                click
                (this.grid as GridComponent).pageSettings.currentPage =
                parseInt((this.textbox as TextBoxComponent).value, 10);
            });
        }
    }
}

```

### MAIN.TS

```

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

```

### Add current page in URL as a query string

The Syncfusion Grid allows you to include the current page information as a query string in the URL. This feature is particularly useful for scenarios where you need to maintain and share the state of the grid's pagination.

To add the current page detail to the URL as a query string in the Syncfusion Grid, you can enable the [enableQueryString](#) property. When this property is set to **true**, it will automatically pass the current

page information as a query string parameter along with the URL when navigating to other pages within the grid.

By enabling the `enableQueryString` property, you can easily copy the URL of the current page and share it with others. When the shared URL is opened, it will load the grid with the exact page that was originally shared.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to enable or disable the addition of the current page to the URL as a query string. When the switch is toggled, the `change` event is triggered and the `enableQueryString` property of the grid is updated accordingly.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { SwitchModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { SwitchComponent } from '@syncfusion/ej2-angular-buttons';
import { orderDetails } from './datasource';
@Component({
  imports: [

    GridModule,
    SwitchModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="padding: 20px 0px 20px 0px">
      <label>Enable/Disable Query String</label>
      <ejs-switch #switch id="switch" [(checked)]="enableQuery"
(change)="toggleQueryString()">
    </ejs-switch>
    </div>
    <ejs-grid #grid [dataSource]='data' allowPaging='true'
[pageSettings]='initialPage'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public initialPage?: object;
  @ViewChild('switch') public switch?: SwitchComponent;
  @ViewChild('grid') public grid?: GridComponent;
```

```

public enableQuery = false;
ngOnInit(): void {
    this.data = orderDetails;
}
toggleQueryString(): void {
    (this.grid as GridComponent).pageSettings.enableQueryString =
this.enableQuery;
}
}

```

## MAIN.TS

```

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

```

## Pager template

The pager template in Syncfusion Grid allows you to customize the appearance and behavior of the pager element, which is used for navigation through different pages of grid data. This feature is particularly useful when you want to use custom elements inside the pager instead of the default elements.

To use the pager template, you need to specify the [pagerTemplate](#) property in your Syncfusion Grid configuration. The `pagerTemplate` property allows you to define a custom template for the pager. Within the template, you can access the [currentPage](#), [pageSize](#), [pageCount](#), **totalPage** and **totalRecordCount** values.

The following example demonstrates how to render a **NumericTextBox** component in the pager using the `pagerTemplate` property:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { data } from './datasource';
import { PageService, GridComponent, PageSettingsModel } from
 '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [

    GridModule,
    NumericTextBoxModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,

```

```

selector: 'app-root',
template: `
  <ejs-grid #grid [dataSource]='data' [allowPaging]='true'
    [pageSettings]='initialPage'>
    <ng-template #pagerTemplate let-data>
      <div class="e-pagertemplate">
        <div class="col-lg-12 control-section">
          <div class="content-wrapper">
            <ejs-numerictextbox format='###.##' step='1' min='1'
max='3' value={{data.currentPage}}
              (change)='change($event)' width="200px"></ejs-
numerictextbox>
          </div>
        </div>
        <div id="totalPages" class="e-pagertemplatemessage"
          style="margin-top:5px;margin-left:30px;border: none;
display: inline-block ">
          <span class="e-pagenomsg">{{data.currentPage}} of
{{data.totalPages}} pages
            ({{data.totalRecordsCount}} items)</span>
        </div>
      </div>
    </ng-template>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {

  @ViewChild('grid')
  public grid?: GridComponent;
  public data: Object[] = [];
  public initialPage?: PageSettingsModel;

  ngOnInit(): void {
    this.data = data;
    this.initialPage = { pageSize: 5 };
  }
  change(args: ChangeEventArgs) {
    this.initialPage = { currentPage: args.value };
  }
}

```

#### APP.TEMPLATE.HTML

```

<ejs-grid #grid [dataSource]='data' [allowPaging]='true'
  [pageSettings]='initialPage'>

```

```

<ng-template #pagerTemplate let-data>
<div class="e-pagertemplate">
  <div class="col-lg-12 control-section">
    <div class="content-wrapper">
      <ejs-numerictextbox format='###.##' step='1' min='1' max='3'
value={{data.currentPage}}
      (change)='change($event)'></ejs-numerictextbox>
    </div>
  </div>
  <div id="totalPages" class="e-pagertemplatemessage"
    style="margin-top:5px;margin-left:30px;border: none; display:
inline-block ">
    <span class="e-pagenomsg">{{data.currentPage}} of
{{data.totalPages}} pages
    ({{data.totalRecordsCount}} items)</span>
  </div>
</div>
</ng-template>
<e-columns>
  <e-column field='OrderID' headerText='Order ID' width=120></e-
column>
  <e-column field='CustomerID' headerText='Customer ID' width=150></e-
column>
  <e-column field='ShipCity' headerText='Ship City' width=150></e-
column>
  <e-column field='ShipName' headerText='Ship Name' width=150></e-
column>
</e-columns>
</ejs-grid>

```

## MAIN.TS

```

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

```

### Pager with page size dropdown

The pager with a page size dropdown in Syncfusion Grid allows you to dynamically change the number of records displayed in the grid. This feature is useful when you want to easily customize the number of records to be shown per page.

To enable the page size Dropdown feature in the Syncfusion Grid, you need to set the [pageSettings.pageSize](#) property to **true** in the grid configuration. This property configuration triggers the rendering of a dropdown list within the pager, allowing you to select the desired page size. The selected page size determines the number of records displayed on each page of the grid.

The following example that demonstrates how to integrate the page size Dropdown feature by configuring the [pageSizes](#) property:

## APP.COMPONENT.TS

```

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

```

```

import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]='true'
height='268px' [pageSettings]='pageSettings'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public pageSettings?: Object;
  ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSizes: true, pageSize: 12 };
  }
}

```

### 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 `pageSizes` property is set to a boolean value like 'true' or 'false,' the page size dropdown defaults to an array of strings containing options such as ['All', '5', '10', '15', '20'].

#### *Customize page size dropdown*

The Syncfusion Grid allows you to customize the default values of the page size dropdown in the pager, allowing you to change the number of records displayed per page. To achieve this, you can define the [pageSizes](#) property as an array of string instead of boolean value.

The following example demonstrate how to customize the default values of the pager dropdown using the `pageSizes` property:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { orderDetails } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' allowPaging='true'
[pageSettings]='initialPage'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public initialPage?: object;
  ngOnInit(): void {
    this.data = orderDetails;
    this.initialPage = { pageSizes: ['5', '10', '15', '20', 'All'], };
  }
}

```

## MAIN.TS

```

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

```

The `pageSizes` property can be configured with either an array of strings or a boolean value.

### How to navigate to particular page

Navigating to a particular page in the Syncfusion Grid is particularly useful when dealing with large datasets. It provides a quick and efficient way to jump to a specific page within the grid.

To achieve page navigation, you can use the [goToPage](#) method provided by Syncfusion Grid. This method allows you to programmatically navigate to a specific page within the grid.

The following example demonstrates how to dynamically navigate to a particular page using the [goToPage](#) method triggered by an external button click based on **TextBox** input:

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { orderDetails } from './datasource';
import { GridComponent, PageSettingsModel } from '@syncfusion/ej2-angular-
grids';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    GridModule,
    TextBoxModule,
    ButtonModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <label style="padding: 30px 17px 0 0">Enter page index:</label>
      <ejs-textbox #textbox width="120"></ejs-textbox>
      <button ejs-button #button id="button"
(created)=clickHandler($event)>click button</button>
    </div>
    <div style="padding:20px 0 0 0">
      <ejs-grid #grid id="PagingGrid" [dataSource]="data"
[allowPaging]="true" height="325">
        <e-columns>
          <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="120">
            </e-column>
          <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
          <e-column field="ShipCity" headerText="Ship City"
width="150"></e-column>
          <e-column field="ShipName" headerText="Ship Name"
width="150"></e-column>
        </e-columns>
      </ejs-grid>
    </div> `
  })
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public pageOptions?: PageSettingsModel;
  @ViewChild('textbox') public textbox?: TextBoxComponent;
  @ViewChild('button') public button?: ButtonComponent;
  ngOnInit(): void {
    this.data = orderDetails;
  }
}

```



```

    }
    clickHandler(args:any): void {
        (this.button as ButtonComponent).element.addEventListener('click',
        (e: MouseEvent) => {
            e.preventDefault(); // Prevent any default behavior of the
            button click
            (this.grid as
            GridComponent).pagerModule.goToPage(parseInt((this.textbox as
            TextBoxComponent).value, 10));
        });
    }
}

```

## 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 get the pager element

You can get pager element in the Syncfusion Grid. This allows you to customize the pager's appearance or behavior to meet the requirements of your application.

[getPager](#)- This method allows you to obtain a reference to the pager element within the Syncfusion Grid. It returns an HTML element representing the pager.

```
`ts
```

```
this.grid.getPager()
```

```
,
```

### Dynamically calculate page size based on element height

You have an option to dynamically calculate the page size of a grid by considering the height of its parent element. This functionality proves invaluable in ensuring that the grid's content remains within the available space, preventing the need for excessive scrolling. It primarily serves the purpose of automatically adjusting the `pageSize` when the height of the grid's parent element changes dynamically. Upon each alteration in the parent element's height, invoking this method will compute the grid's `pageSize` and present the current page records accordingly. This feature effectively addresses situations where a static `pageSize` value does not cater to the varying heights of different parent elements, preventing any unwanted empty spaces within the grid.

To achieve page size calculation based on an element's height in the Grid, you can utilize the [calculatePageSizeByParentHeight](#) method. This method calculates the page size based on the height of the parent element.

The following example demonstrates how to calculate the page size based on element height using the `calculatePageSizeByParentHeight` method triggered by a change event based on the **NumericTextBox** input:

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit, ViewChild } from '@angular/core';
import { orderDetails } from './datasource';
import { ChangeEventArgs, NumericTextBoxComponent } from '@syncfusion/ej2-
angular-inputs';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    NumericTextBoxModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="padding:0 0 20px 0">
      <label style="padding: 30px 17px 0 0">Select page size:</label>
      <ejs-numerictextbox #numericTextbox placeholder='select
container height'
        format='###.##' min=150 step="50"
        (change)='calculatePageSize($event)'
        width="200px"></ejs-numerictextbox>
    </div>
    <ejs-grid #grid [dataSource]="data" [allowPaging]="true" >
      <e-columns>
        <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="90">
          </e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="120"></e-column>
        <e-column field="Freight" headerText="Freight" textAlign='Right'
format='C2'
          width="90"></e-column>
        <e-column field="OrderDate" headerText="Order Date"
textAlign='Right'
          format='yMd' width="120"></e-column>
      </e-columns>
    </ejs-grid>`
  })
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') grid?: GridComponent;
  @ViewChild('numericTextbox') public numerictextbox?:
NumericTextBoxComponent;
  ngOnInit(): void {
    this.data = orderDetails;
  }
  calculatePageSize({ value }: ChangeEventArgs) {
    (this.grid as GridComponent).pageSettings.pageSize = (this.grid as
GridComponent).calculatePageSizeByParentHeight((value as
number).toString());
  }
}

```

```
}

```

**MAIN.TS**

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

**Render pager at the top of the grid**

The Grid component provides built-in support for rendering a pager at the bottom of the grid by default. However, in certain scenarios, you might want to display the pager at the top of the grid. This can be achieved by utilizing the [dataBound](#) event. This event is triggered when the Grid completes rendering its data. By handling this event, you can customize the rendering of the pager and move it to the top of the Grid.

Here's an example that demonstrates how to render the pager at the top of the grid using the `dataBound` event:

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='pagerAtTop' [dataSource]='data'
[allowPaging]='true' (dataBound)='dataBound()'
height='268px' [pageSettings]='pageSettings'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
    <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
    <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageSettings?: object;
  @ViewChild('grid')
```

```

public grid?: GridComponent;
public toolbar?: ToolbarItems[];
public initialGridLoad: boolean = true;
ngOnInit(): void {
    this.data = data;
    this.pageSettings = { pageSizes: true, pageSize: 12 };
}
dataBound() {
    if (this.initialGridLoad) {
        this.initialGridLoad = false;
        const pager = document.getElementsByClassName('e-gridpager');
        let topElement;
        if ((this.grid as any).allowGrouping || (this.grid as
any).toolbar) {
            topElement = (this.grid as any).allowGrouping ?
document.getElementsByClassName('e-groupdroparea') :
                document.getElementsByClassName('e-toolbar');
        } else {
            topElement = document.getElementsByClassName('e-
gridheader');
        }
        (this.grid as any).element.insertBefore(pager[0],
topElement[0]);
    }
}
}

```

### MAIN.TS

```

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

```

During the paging action, the pager component triggers the below three events.

- \* The [created](#) event triggers when Pager is created.
- \* The [click](#) event triggers when the numeric items in the pager is clicked.
- \* The [dropDownChanged](#) event triggers when pageSize DropDownList value is selected.

### Pager events

The Syncfusion Grid component triggers two pager events during paging actions:

[actionBegin](#)- This event triggered before any paging action (such as changing the page, changing the page size and etc) is initiated. You can use this event to customize or control the behavior of paging actions.

[actionComplete](#)- This event triggered after a pager action is completed. It provides information about the action, such as the new page number, page size, and the total number of records. You can use this event to perform actions or update the UI after the operation has been executed.

The following example that example demonstrates how to use these events to display notification messages to indicate the current and next page during paging actions in the Syncfusion Angular Grid:

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService, EditService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { PageEventArgs } from '@syncfusion/ej2-grids';
import { orderDetails } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService, ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `
    <p id="message1">{{ message1 }}</p>
    <p id="message">{{ message }}</p>
    <ejs-grid #grid [dataSource]="data" allowPaging="true"
      (actionBegin)="onActionBegin($event)"
      (actionComplete)="onActionComplete($event)"
      [pageSettings]="initialPage">
      <e-columns>
        <e-column field="OrderID" headerText="Order ID"
textAlign="Right" width="90"></e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="120"></e-column>
        <e-column field="Freight" headerText="Freight"
textAlign="Right" format="C2" width="90"></e-column>
        <e-column field="OrderDate" headerText="Order Date"
textAlign="Right" format="yMd" width="120"></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public initialPage?: object;
  @ViewChild('grid') grid?: GridComponent;
  public message?: string;
  public message1?: string;
  ngOnInit(): void {
    this.data = orderDetails;
    this.initialPage = { pageSize: 5 };
  }
  onActionBegin({requestType, currentPage, previousPage}: PageEventArgs) {
    if (requestType === 'paging') {
      this.message = (currentPage as string) > (previousPage as
string)
        ? `You are going to switch to page ${parseInt((currentPage
as string), 10) + 1}`
        : `You are going to switch to page ${previousPage}`;
    }
  }
  onActionComplete(args: PageEventArgs) {

```

```

        if (args.requestType === 'paging') {
            this.message1 = 'Now you are in page ' + args.currentPage;
        }
    }
}

```

### 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

- [Group with Paging](#)
- [How to change loading indicator in Angular Grid](#)

### Selection in Angular Grid component

Selection in the Grid component allows you to interactively select specific cells, rows, or columns within the grid. This selection can be done through mouse clicks or arrow keys (up, down, left, and right) or touch. This feature is useful when you want to highlight, manipulate, or perform actions on specific cells, rows, or columns within the Grid.

To disable selection in the Grid, set the [allowSelection](#) to **false**.

The grid supports two types of selection that can be set by using the [selectionSettings.type](#). They are:

- **Single** - The **Single** value is set by default. Allows you to select only a single row or cell or column.
- **Multiple** - Allows you to select multiple rows or cells or columns.

To perform the multi-selection, press and hold CTRL key and click the desired rows or cells or columns.

To select range of rows or cells or columns, press and hold the SHIFT key and click the rows or cells or columns.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, EditService, ToolbarService, FilterService } from '@syncfusion/ej2-angular-grids'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, SelectionSettingsModel, SelectionType } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
    imports: [

        GridModule,

```

```

        DropDownListModule
    ],
    providers: [EditService, ToolbarService, PageService, FilterService],
    standalone: true,
    selector: 'app-root',
    template: `
        <div style="display: flex">
            <label style="padding: 30px 17px 0 0">Choose selection type:</label>
            <ejs-dropdownlist style="padding: 26px 0 0 0" index="0" width="150"
            [dataSource]="dropdownData" (change)="valueChange($event)">
            </ejs-dropdownlist>
        </div>
        <div style="padding: 20px 0px 0px 0px">
            <ejs-grid #grid [dataSource]="data"
            [selectionSettings]="selectionOptions"
            height="315px">
                <e-columns>
                    <e-column field="OrderID" headerText="Order ID"
                    textAlign="Right"
                    width="120"></e-column>
                    <e-column field="CustomerID" headerText="Customer ID"
                    width="150">
                        </e-column>
                    <e-column field="ShipCity" headerText="Ship City"
                    width="150"></e-column>
                    <e-column field="ShipName" headerText="Ship Name"
                    width="150"></e-column>
                </e-columns>
            </ejs-grid>
        </div>`
    })
    export class AppComponent implements OnInit {
        public data?: Object[];
        @ViewChild('grid') public grid?: GridComponent;
        public selectionOptions?: SelectionSettingsModel;
        public dropdownData: Object[] = [
            { text: 'Single', value: 'Single' },
            { text: 'Multiple', value: 'Multiple' }
        ];
        ngOnInit(): void {
            this.data = data;
        }
        valueChange(args: ChangeEventArgs): void {
            ((this.grid as GridComponent).selectionSettings.type as SelectionType) =
            (args.value as SelectionType);
        }
    }
}

```

## 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 Grid will be rendered with only the basic features such as Grid rendering and selection. The default module, including the `SelectionService`, is automatically loaded. Therefore, there is no need to inject the `SelectionService` module separately as it will be injected by default.

### Selection mode

The selection mode feature allows you to choose between different modes for selecting rows or cells or both within the Grid based on your specific requirements. This feature is particularly useful when you want to highlight and manipulate specific rows or cells in the Grid.

To enable selection mode, you can set the [selectionSettings.mode](#) property. The Grid component supports three types of selection modes:

- **Row** - The row value is set by default. Allows you to select rows only.
- **Cell** - Allows you to select cells only.
- **Both** - Allows you to select rows and cells at the same time.

The following example, demonstrates how to dynamically enable and change the `selectionSettings.mode` using the `DropDownList` component:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, EditService, ToolbarService, FilterService } from '@syncfusion/ej2-angular-grids'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListModule
  ],
  providers: [EditService, ToolbarService, PageService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 30px 17px 0 0">Choose selection
mode:</label>
      <ejs-dropdownlist style="padding: 26px 0 0 0" index="0"
width="150"
        [dataSource]="dropdownData" (change)="valueChange($event)">
      </ejs-dropdownlist>
    </div>
    <div style="padding: 20px 0px 0px 0px">
      <ejs-grid #grid [dataSource]="data" height="315px">
        <e-columns>
          <e-column field="OrderID" headerText="Order ID"
textAlign="Right"
            width="120"></e-column>
```



```

        <e-column field="CustomerID" headerText="Customer ID"
width="150">
        </e-column>
        <e-column field="ShipCity" headerText="Ship City"
width="150"></e-column>
        <e-column field="ShipName" headerText="Ship Name"
width="150"></e-column>
    </e-columns>
</ejs-grid>
</div>`
}))
export class AppComponent implements OnInit {
    public data?: Object[];
    @ViewChild('grid') public grid?: GridComponent;
    public selectionOptions?: SelectionSettingsModel;
    public dropdownData: Object[] = [
        { text: 'Row', value: 'Row' },
        { text: 'Cell', value: 'Cell' },
        { text: 'Both', value: 'Both' }
    ];
    ngOnInit(): void {
        this.data = data;
    }
    valueChange(args: ChangeEventArgs): void {
        ((this.grid as GridComponent).selectionSettings.mode as SelectionMode) =
        (args.value as SelectionMode);
    }
}

```

## MAIN.TS

```

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

```

## Touch interaction

The touch interaction feature in Grid allows you to easily interact with the grid on touch screen devices. This feature is particularly useful for improving the user experience on mobile devices and tablets, making it easier to navigate and interact with the grid's content using touch gestures.

### Single Row Selection

When you tap on a grid row using a touch screen, the tapped row is automatically selected. This provides a straightforward way to select single rows with a touch interface.

### Multi-Row Selection

To select multiple rows in the grid, you can utilize the multi-row selection feature. When you tap on a row, a popup is displayed, indicating the option for multi-row selection. You can tap on the popup, and then proceed to tap on the desired rows that you want to select. This allows you to select and interact with multiple rows simultaneously, as shown in the following image:



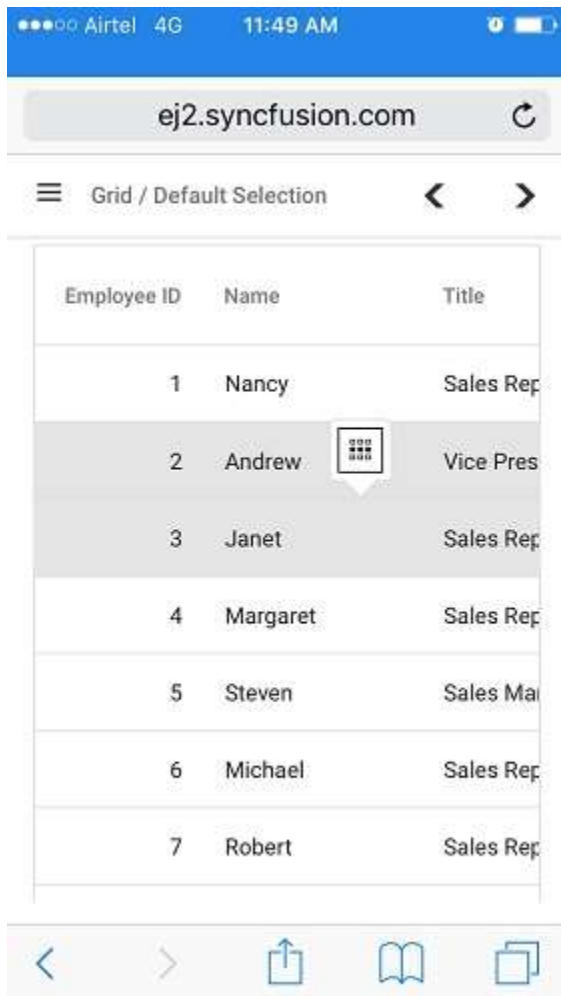
### Multi-Row or Cell Selection

In addition to selecting multiple rows, you can also perform multi-row or cell selection in the grid. By tapping on the popup, you can choose the option for multi-row or cell selection. Once selected, you can then tap on the desired rows or cells to make the selection, as shown in the following image:



For multi-selection, it requires the selection [type](#) to be **Multiple**.

The following screenshot represents a Grid touch selection in the device.



### Toggle selection

The toggle selection feature in the Grid component allows you to easily select and unselect specific rows, cells, or columns. With toggle selection enabled, you can easily switch the selection state of an item by clicking on it. This means that if you click on a selected row, cell, or column, it will become unselected, and vice versa.

To enable the toggle selection feature, you need to set the [selectionSettings.enableToggle](#) property to **true**.

The following example demonstrates how to enable the toggle selection for both cells and rows in a Grid using the [selectionSettings.enableToggle](#) property.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { EditService, ToolbarService, FilterService } from '@syncfusion/ej2-angular-grids'
import { SwitchModule } from '@syncfusion/ej2-angular-buttons'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource'; // Replace with your data source
import { GridComponent, SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { SwitchComponent } from '@syncfusion/ej2-angular-buttons';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [

    GridModule,
    SwitchModule,
    DropDownListModule
  ],
  providers: [EditService, ToolbarService, PageService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 30px 17px 0 0">Choose cell selection
mode:</label>
      <ejs-dropdownlist style="padding: 26px 0 0 0" index="0" width="150"
[dataSource]="dropdownData" (change)="valueChange($event)">
      </ejs-dropdownlist>
    </div>
    <div style="padding: 20px 0px 20px 0px">
      <label>Enable/Disable Toggle selection</label>
      <ejs-switch #switch id="switch" [checked]="true"
(change)="toggleColumnSelection($event)">
      </ejs-switch>
    </div>
    <div style="padding: 20px 0px 0px 0px">
      <ejs-grid #grid [dataSource]="data"
[selectionSettings]="selectionOptions"
height="315px">
        <e-columns>
          <e-column field="OrderID" headerText="Order ID"
textAlign="Right"
width="120"></e-column>
          <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
          <e-column field="ShipCity" headerText="Ship City"
width="150"></e-column>

```

```

        <e-column field="ShipName" headerText="Ship Name"
width="150"></e-column>
    </e-columns>
</ejs-grid>
</div>
))
export class AppComponent implements OnInit {
    public data?: object[];
    public enableToggleSelection = true;
    @ViewChild('grid') public grid?: GridComponent;
    @ViewChild('switch') public switch?: SwitchComponent;
    public selectionOptions?: SelectionSettingsModel;
    public dropdownData: Object[] = [
        { text: 'Row', value: 'Row' },
        { text: 'Cell', value: 'Cell' },
        { text: 'Both', value: 'Both' },
    ];
    ngOnInit(): void {
        this.data = data;
        this.selectionOptions = { type: 'Multiple' };
    }
    toggleColumnSelection(args: CustomChangeEventArgs): void {
        (this.grid as GridComponent).selectionSettings.enableToggle =
args.checked;
    }
    valueChange(args: ChangeEventArgs): void {
        ((this.grid as GridComponent).selectionSettings.mode as SelectionMode) =
(args.value as SelectionMode);
    }
}
interface CustomChangeEventArgs extends ChangeEventArgs {
    checked: boolean;
}

```

### MAIN.TS

```

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

```

\* If multi selection is enabled, then first click on any selected row (without pressing Ctrl key), it will clear the multi selection and in second click on the same row, it will be unselected.

\* Toggle selection is a feature that can be applied to all types of selections. When the `checkboxOnly` property is set to `true`, it restricts the ability to select or deselect rows or cells by clicking on them.

### Clear all selection programmatically

The clear selection programmatically feature is particularly useful when you need to remove the selected rows or cells or columns from the Grid component.

To clear the selection in the component programmatically, you can use the [clearSelection](#) method.

In the following example, it demonstrates how to clear all selection by calling the [clearSelection](#) method in the button click event.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { EditService, ToolbarService, FilterService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, SelectionSettingsModel, PageSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [EditService, ToolbarService, PageService, FilterService],
  standalone: true,
  selector: 'app-root',
  template:
    `<div style="padding: 20px 0px 0px 0px">
      <button ej2-button (click)='click()'>Clear Selection</button>
    </div>
    <div style="padding: 20px 0px 0px 0px">
      <ejs-grid #grid [dataSource]='data' allowPaging=true
        [selectionSettings]='selectionOptions' [pageSettings]='pageOptions'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
          width=120></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
          <e-column field='ShipCountry' headerText='Ship Country'
width=130></e-column>
          <e-column field='Freight' headerText='Freight' format= 'C2'
width=100>
            </e-column>
          </e-columns>
        </ejs-grid>
      </div>`
  })
export class AppComponent implements OnInit {
  public data?: Object[];
  public selectionOptions?: SelectionSettingsModel;
  public pageOptions?: PageSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.selectionOptions = { mode: 'Both' ,allowColumnSelection:
true,type: 'Multiple' };
    this.pageOptions = { pageSize: 5 };
  }
}
```

```

    }
    click(): void{
        (this.grid as GridComponent).clearSelection();
    }
}

```

### MAIN.TS

```

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

```

In **Both** mode, if you call **clearCellSelection** first, it will clear cell selections, and then if you call **clearRowSelection**, it will clear row selections. The order of method calls determines which type of selection is cleared first.

To remove a specific selection in a row, cell, or column, utilize the following methods: **clearRowSelection** for clearing row selections, **clearCellSelection** for clearing cell selections, and **clearColumnSelection** for clearing column selections.

### Persist selection

Persist selection feature in the Grid allows you to retain the selected items even after data manipulation or refreshing the grid. This feature is useful when you want to keep track of the selected items across different grid operations.

To enable persist selection, set the [selectionSettings.persistSelection](#) property to **true**.

\* While using persist selection feature, at least one column in your grid should be enabled as a primary key. This ensures that the grid can identify and persist the selected items correctly.

\* The **persistSelection** feature is not supported for cell selections in the Syncfusion Angular Grid component

In the following example, it demonstrates how to enable the persist selection feature for both rows and columns using the **selectionSettings.persistSelection** property :

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { EditService, ToolbarService, FilterService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel, SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [EditService, ToolbarService, PageService, FilterService],
  standalone: true,

```

```

    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowPaging]='true'
height='315px'
    [selectionSettings]='selectionOptions'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
isPrimaryKey='true'
        textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public selectionOptions?: SelectionSettingsModel;
    public pageOptions?: PageSettingsModel;
    ngOnInit(): void {
      this.data = data;
      this.selectionOptions = { type: 'Multiple', mode: 'Both',
persistSelection: true, allowColumnSelection: true };
      this.pageOptions = { pageSize: 5 };
    }
  }

```

## 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

- [Animate the Grid selected row in Angular Grid](#)
- [How can I disable the row selection on context menu click in Angular Grid](#)
- [How to retrieve the selected records in the Grid in Angular Grid](#)
- [How to prevent tab to focus on a cell when inside a grid in Angular Grid](#)
- [How to select the first row of the Grid, after the grid refreshed in Angular Grid](#)
- [How to select the multiple row at initial render in Angular Grid](#)
- [How to cancel the selection of first record while adding a new record in Angular Grid](#)

## Aggregates in Angular Grid component

The Aggregates feature in the Syncfusion angular Grid component allows you to display aggregate values in the footer, group footer, and group caption of the grid. With this feature, you can easily perform calculations on specific columns and show summary information. This feature can be configured using the **e-aggregates** directive. To represent an aggregate column, you need to specify the minimum required properties, such as [field](#) and [type](#).

To use aggregate feature, you need to inject the **AggregateService** module into the **@NgModule.providers** section.

### Displaying aggregate values

By default, the aggregate values are displayed in the footer, group, and caption cells of the grid. However, you can choose to display the aggregate value in any of these cells by using the following properties:

- **footerTemplate**: Use this property to display the aggregate value in the footer cell. You can define a custom template to format the aggregate value as per your requirements.
- **groupFooterTemplate**: Use this property to display the aggregate value in the group footer cell. Similar to the footerTemplate, you can provide a custom template to format the aggregate value.
- **groupCaptionTemplate**: Use this property to display the aggregate value in the group caption cell. You can define a custom template to format the aggregate value.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { AggregateService, GroupService, PageService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [AggregateService, GroupService, PageService],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'app.template.html'
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public groupOptions: GroupSettingsModel = { showDropArea: false,
  columns: ['ShipCountry'] };
  ngOnInit(): void {
    this.data = data;
  }
}
```

### APP.TEMPLATE.HTML

```
<ejs-grid [dataSource]='data' height='290px' [allowGrouping]='true'
[allowPaging]='true' [groupSettings]='groupOptions'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID' textAlign='right'
width=120></e-column>
```



```

    <e-column field='CustomerID' headerText='Customer ID' width=150></e-
column>
    <e-column field='OrderDate' headerText='Order Date' format='yMd'
width=120></e-column>
    <e-column field='Freight' format='C2' width=150></e-column>
    <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
  </e-columns>
  <e-aggregates>
    <e-aggregate>
      <e-columns>
        <e-column field='Freight' type='sum'>
          <ng-template #groupFooterTemplate let-data>Sum:
{{data.sum}}</ng-template>
        </e-column>
        <e-column field='Freight' type='max'>
          <ng-template #groupCaptionTemplate let-data>Max:
{{data.max}}</ng-template>
        </e-column>
      </e-columns>
    </e-aggregate>
  </e-aggregates>
</ejs-grid>

```

### MAIN.TS

```

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

```

\* When using local data, the total summary is calculated based on the entire dataset available in the grid. The aggregate values will reflect calculations across all the rows in the grid.

\* When working with remote data, the total summary is calculated based on the current page records. This means that if you have enabled pagination and are displaying data in pages, the aggregate values in the footer will represent calculations only for the visible page.

### Built-in aggregate types

The Syncfusion Angular Grid component provides several built-in aggregate types that can be specified in the [type](#) property to configure an aggregate column.

The available built-in aggregate types are:

- **Sum:** Calculates the sum of the values in the column.
- **Average:** Calculates the average of the values in the column.
- **Min:** Finds the minimum value in the column.
- **Max:** Finds the maximum value in the column.
- **Count:** Counts the number of values in the column.
- **TrueCount:** Counts the number of true values in the column.
- **FalseCount:** Counts the number of false values in the column.

Here is an example that demonstrates how to use built-in aggregates types in the Syncfusion Grid:

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { AggregateService, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { GroupSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [AggregateService, GroupService],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'app.template.html'
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public groupOptions: GroupSettingsModel = { showDropArea: false,
columns: ['ShipCountry'] };
  ngOnInit(): void {
    this.data = data;
  }
}

```

**APP.TEMPLATE.HTML**

```

<ejs-grid [dataSource]="data" height="290px" [allowGrouping]="true"
[groupSettings]="groupOptions">
  <e-columns>
    <e-column field="OrderID" headerText="Order ID" textAlign="right"
width="120"></e-column>
    <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
    <e-column field="OrderDate" headerText="Order Date" format="yMd"
type="date" width="120"></e-column>
    <e-column field="ShippedDate" headerText="Shipped Date" format="yMd"
type="date" width="120"></e-column>
    <e-column field="Freight" format="C2" width="150"></e-column>
    <e-column field="isVerified" headerText="Verified" width="150"
type="boolean"></e-column>
    <e-column field="ShipCity" headerText="ShipCity" width="150"></e-
column>
    <e-column field="ShipCountry" headerText="Ship Country"
width="150"></e-column>
  </e-columns>
  <e-aggregates>
    <e-aggregate>
      <e-columns>
        <e-column field="Freight" type="max">
          <ng-template #footerTemplate let-data>Max: {{ data.max
}}</ng-template>

```

```

        </e-column>
        <e-column field="ShippedDate" type="max">
            <ng-template #footerTemplate let-data>Max: {{ data.max |
date: 'dd/MM/yyyy' }}</ng-template>
        </e-column>
        <e-column field="OrderDate" type="min">
            <ng-template #footerTemplate let-data>Min: {{ data.min |
date: 'dd/MM/yyyy' }}</ng-template>
        </e-column>
        <e-column field="isVerified" type="truecount">
            <ng-template #footerTemplate let-data>TrueCount: {{
data.truecount }}</ng-template>
        </e-column>
    </e-columns>
</e-aggregate>
</e-aggregates>
</ejs-grid>

```

### MAIN.TS

```

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

```

### Multiple aggregates for a column

Multiple aggregates for a column allows you to calculate and display different summary values simultaneously for a single column in a grid. Normally, a column is associated with a single aggregate function, such as sum, average, count and etc., which provides a single summary value for the entire column.

However, in scenarios where you need to display multiple summary values for the same column, multiple aggregates come into play. This feature enables you to calculate and display various aggregate values, such as sum, average, minimum, maximum, or custom calculations, concurrently for a specific column.

You can use multiple aggregates for a single column in the Syncfusion Angular Grid by specifying the aggregate [type](#) as an array.

Here's an example of how to use multiple aggregates in the Syncfusion Angular Grid:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { AggregateService, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],

```

```

providers: [AggregateService, GroupService],
standalone: true,
  selector: 'app-root',
  templateUrl: 'app.template.html'
}))
export class AppComponent implements OnInit {
  public data?: Object[];
  public aggregates = [
    {
      columns: [
        {
          type: ['Sum', 'Max', 'Min'],
          field: 'Freight',
          columnName: 'Freight',
          format: 'C2',
          footerTemplate: 'Sum: ${Sum}, Min:${Min}, Max:${Max}',
        },
      ],
    },
  ];
  ngOnInit(): void {
    this.data = data;
  }
}

```

#### APP.TEMPLATE.HTML

```

<ejs-grid [dataSource]="data" height="290px" [allowPaging]="true"
[aggregates]="aggregates">
  <e-columns>
    <e-column field="OrderID" headerText="Order ID" textAlign="right"
width="120"></e-column>
    <e-column field="CustomerID" headerText="Customer ID"
width="150"></e-column>
    <e-column field="Freight" format="C2" width="150"></e-column>
    <e-column field="ShipCountry" headerText="Ship Country"
width="150"></e-column>
  </e-columns>
</ejs-grid>

```

#### 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

- [Tooltip for aggregate footer in Angular Grid](#)
- [How to export aggregate footer and apply outer border on excel data in Angular Grid](#)

## Print in Angular Grid component

The printing feature in Syncfusion Grid allows you to easily generate and print a representation of the grid's content for better offline accessibility and documentation. You can enable this feature using either the grid's toolbar or the programmatically available `print` method.

To add the printing option to the grid's toolbar, simply include the `toolbar` property in your grid configuration and add the **Print** as toolbar item. This will allow you to directly initiate the printing process while click on the Print item from the toolbar.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [toolbar]='toolbarOptions'
height='272px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['Print'];
  }
}
```

### MAIN.TS

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

### Page setup

When printing a webpage, some print options, such as layout, paper size, and margin settings, cannot be configured through JavaScript code. Instead, you need to customize these settings using the browser's page setup dialog. Below are links to the page setup guides for popular web browsers:

- [Chrome](#)
- [Firefox](#)
- [Safari](#)
- [IE](#)

### Print by external button

You can print the grid's content using an external button by utilizing the [print](#) method. This method allows you to trigger the printing process programmatically.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [PageService],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button cssClass='e-primary' id='print'
(click)='print()'>Print</button>
    <ejs-grid #grid [dataSource]='data' height='280px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid') public gridObj?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  print() {
    (this.gridObj as GridComponent).print();
  }
}
```

```
}
}
```

## MAIN.TS

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

## Print visible Page

By default, the Syncfusion Angular Grid prints all the pages of the grid. The [printMode](#) property within the grid grants you control over the printing process. However, if you want to print only the current visible page, you can achieve this by setting the [printMode](#) property to **CurrentPage**.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PageService } from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, PageSettingsModel } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [

    GridModule,
    DropDownListAllModule
  ],
  providers: [ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<div style="display:flex;">
<label style="padding: 10px 10px 26px 0">Select Print Mode </label>
<ejs-dropdownlist
  style="margin-top:5px"
  id="value"
  #dropdown
  index="0"
  width="120"
  [dataSource]="dropdownlist"
  (change)="onChange($event)"
></ejs-dropdownlist></div>
      <ejs-grid [dataSource]='data' [printMode]='printMode'
[allowPaging]='true'
      [pageSettings]='pageOptions' [toolbar]='toolbarOptions'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
```

```

        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        public pageOptions?: PageSettingsModel;
        public printMode: string = 'CurrentPage';
        public dropdownlist: string[] = ['AllPages', 'CurrentPage'];
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['Print'];
            this.pageOptions = { pageSize: 6 };
        }
        onChange(args: ChangeEventArgs): void {
            this.printMode = 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));

```

### Print only selected records

By default, the Syncfusion Angular Grid prints all the data bound to its dataSource. However, there might be cases where you want to print only the selected records from the grid. The Angular Grid provides an option to achieve this by binding to the [beforePrint](#) event, where you can replace the rows of the printing grid with the selected rows.

Below is an example code that demonstrates how to print only the selected records from the Angular Grid:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, PrintEventArgs, ToolbarItems, PageSettingsModel, SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { createElement } from '@syncfusion/ej2-base';
@Component({
    imports: [
        GridModule
    ],

```



```

providers: [ToolbarService, PageService],
standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='grid' [dataSource]='data'
[allowPaging]='true'
    [pageSettings]='pageOptions'
[selectionSettings]="selectionSettings" (beforePrint)='beforePrint($event)'
[toolbar]='toolbarOptions'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
))
export class AppComponent implements OnInit {
  @ViewChild('grid', { static: true })
  public grid?: GridComponent;
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  public pageOptions?: PageSettingsModel;
  public selectionSettings?: SelectionSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['Print'];
    this.pageOptions = { pageSize: 6 };
    this.selectionSettings = { type: 'Multiple' }
  }
  beforePrint(e: PrintEventArgs): void {
    var rows = (this.grid as GridComponent).getSelectedRows();
    if (rows.length) {
      (e.element as
CustomElement).ej2_instances[0].getContent().querySelector('tbody').remove()
;
      var tbody = createElement('tbody');
      rows = [...rows];
      for (var r = 0; r < rows.length; r++) {
        tbody.appendChild(rows[r].cloneNode(true));
      }
      (e.element as
CustomElement).ej2_instances[0].getContentTable().appendChild(tbody);
    }
  }
}
interface CustomElement extends Element {
  ej2_instances: any[];
}

```

## 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 hierarchy grid

The Syncfusion Angular Grid allows you to print hierarchy grids, which consist of a parent grid and its child grids. By default, when you print a hierarchy grid, it includes the parent grid and expanded child grids only. However, you can customize the print behavior using the [hierarchyPrintMode](#) property.

The `hierarchyPrintMode` property in the Angular Grid lets you control the printing behavior for hierarchy grids. You can choose from three options:

Mode	Behavior
Expanded	Prints the parent grid with expanded child grids.
All	Prints the parent grid with all the child grids, whether expanded or collapsed.
None	Prints the parent grid alone.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule
 } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns'
import { DetailRowService, GridModel, ToolbarService, GridComponent } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<div class='container'><h4>Select Mode</h4>
    <ejs-dropdownlist style="margin: 14px; width:250px; " #sample
    [dataSource]='dropdownData' (change)='onModeChange($event)'
    popupHeight='220px'></ejs-dropdownlist></div>
    <ejs-grid #grid [dataSource]='parentData' height='265px'
    [childGrid]='childGrid' [toolbar]='["Print"]'
    [hierarchyPrintMode]='hierarchyPrintMode'>
      <e-columns>
```

```

        <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=120></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City'
width=150></e-column>
    </e-columns>
</ejs-grid>
    providers: [DetailRowService, ToolbarService]
})
export class AppComponent implements OnInit {
    public parentData?: object[];
    public dropdownData?: string[] = ['All', 'Expanded', 'None'];
    public hierarchyPrintMode: string = 'All';
    public childGrid: GridModel = {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
            { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 120 },
            { field: 'CustomerID', headerText: 'Customer ID', width: 150 },
            { field: 'ShipCity', headerText: 'Ship City', width: 150 },
            { field: 'ShipName', headerText: 'Ship Name', width: 150 }
        ],
    };
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.parentData = employeeData;
    }
    onModeChange(args: ChangeEventArgs): void {
        this.hierarchyPrintMode = args.value 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));

```

### Print the master detail grid

The Syncfusion Angular Grid provides the option to visualize details of a record in another grid in a master-detail manner. By default, when you print a master-detail grid, only the master grid is included in the print output. However, you can customize the print behavior to include both the master and detail grids using the `beforePrint` event of the grid.

The `beforePrint` event in the Angular Grid is triggered before the actual printing process begins. You can handle this event to customize the print output. By adding the detail grid to the `element` argument of the `beforePrint` event, you can ensure that both the master and detail grids are printed on the page.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, customerData } from './datasource';
import { GridComponent, ToolbarItems, RowSelectEventArgs, PrintEventArgs }
from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #mastergrid id="mastergrid"
[dataSource]='masterdata' [selectedRowIndex]='1' [toolbar]='toolbar'
(rowSelected)="onRowSelected($event)" (beforePrint)="beforePrint($event)">
  <e-columns>
    <e-column field='ContactName' headerText='Customer Name'
width='150'></e-column>
    <e-column field='CompanyName' headerText='Company Name'
width='150'></e-column>
    <e-column field='Address' headerText='Address' width='150'></e-
column>
    <e-column field='Country' headerText='Country' width='130'></e-
column>
  </e-columns>
</ejs-grid>
<div class='e-statustext' style="margin:8px">Showing orders of Customer:
<b id="key"></b></div>

    <ejs-grid #detailgrid id="detailgrid" [dataSource]='data'
[allowSelection]='false'>
    <e-columns>
    <e-column field='OrderID' headerText='Order ID' width='100'></e-
column>
    <e-column field='Freight' headerText='Freight' format='C2'
width='100' type='number'></e-column>
    <e-column field='ShipName' headerText="Ship Name"
width='150'></e-column>
    <e-column field='ShipCountry' headerText="Ship Country"
width='150'></e-column>
    <e-column field='ShipAddress' headerText="Ship Address"
width='150'></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public names?: string[] = ['AROUT', 'BERGS', 'BLONP', 'CHOPS', 'ERNSH'];
  public toolbar?: ToolbarItems[];
  @ViewChild('mastergrid') public mastergrid?: GridComponent;
  @ViewChild('detailgrid') public detailgrid?: GridComponent;
  public masterdata?: Object[];
  public data: object[] = data;

```

```

ngOnInit(): void {
    this.masterdata = customerData.filter((e: object) => (this.names as
string[]).indexOf((e as ItemType).CustomerID) !== -1);
    this.toolbar = ['Print'];
}
public onRowSelected(args: RowSelectEventArgs): void {
    let selectedRecord: object = args.data as object;
    (this.detailgrid as GridComponent).dataSource = data.filter((record:
object) => (record as ItemType).CustomerName === (selectedRecord as
ItemType).ContactName).slice(0, 5);
    (document.getElementById('key') as Element).innerHTML =
(selectedRecord as ItemType).ContactName;
}
public beforePrint(args: PrintEventArgs): void {
    let customElement = document.createElement('div');
    customElement.innerHTML = document.getElementsByClassName('e-
statustext')[0].innerHTML + (this.detailgrid as
GridComponent).element.innerHTML;
    customElement.appendChild(document.createElement('br'));
    (args.element as Element).append(customElement);
}
}
interface ItemType{
    CustomerName:string,
    CustomerID:string,
    ContactName:string
}

```

### MAIN.TS

```

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

```

### Print large number of columns

When printing a grid with a large number of columns, the browser's default page size (usually A4) might not be sufficient to display all the columns properly. As a result, the browser's print preview may automatically hide the overflowed content, leading to a cut-off appearance.

To show a large number of columns when printing, you can adjust the scale option from the print option panel based on your content size. This will allow you to fit the entire grid content within the printable area.



### Show or hide columns while printing

In the Syncfusion Angular Grid, you have the flexibility to control the visibility of columns during the printing process. You can dynamically show or hide specific columns using the [toolbarClick](#) and [printComplete](#) events while printing. This capability enhances your control over which columns are included in the printed output, allowing you to tailor the printed grid to your specific needs.

In the [toolbarClick](#) event, you can show or hide columns by setting [column.visible](#) property to **true** or **false** respectively.

In the [printComplete](#) event, the column visibility state is reset back to its original configuration.

Here's a code example that demonstrates how to show a hidden column (CustomerID) and hide a visible column (ShipCity) during printing and then reset their visibility after printing:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ClickEventArgs } from '@syncfusion/ej2-inputs'
import { ToolbarItems, PageSettingsModel, GridComponent, Column } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
(toolbarClick)='toolbarClick($event)' (printComplete)='printComplete()'
[allowPaging]='true' [pageSettings]='pageOptions'
[toolbar]='toolbarOptions'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>`
})
export class AppComponent implements OnInit {
  toolbarOptions: ToolbarItems = [
    { text: 'Print', iconClass: 'fa-print', click: this.toolbarClick }
  ];
  pageOptions: PageSettingsModel = {
    pageLength: 10,
    totalRecords: 100
  };
  ngOnInit(): void {
    this.grid = document.getElementById('Grid');
  }
  toolbarClick(e: ClickEventArgs): void {
    const column = this.grid.getColumn('ShipCity');
    column.visible = false;
    const column = this.grid.getColumn('CustomerID');
    column.visible = true;
  }
  printComplete(): void {
    const column = this.grid.getColumn('ShipCity');
    column.visible = true;
    const column = this.grid.getColumn('CustomerID');
    column.visible = false;
  }
}
```

```

        <e-column field='CustomerID' headerText='Customer ID'
[visible]= 'false' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        public pageOptions?: PageSettingsModel;
        @ViewChild('grid')
        public grid?: GridComponent;
        toolbarClick(args: ClickEventArgs) {
            if (args.item.id == 'Grid_print') {
                for (const columns of (this.grid as GridComponent).columns) {
                    if ((columns as Column).field === 'CustomerID') {
                        (columns as Column).visible = true;
                    } else if ((columns as Column).field === 'ShipCity') {
                        (columns as Column).visible = false;
                    }
                }
            }
        }
        printComplete() {
            for (const columns of (this.grid as GridComponent).columns) {
                if ((columns as Column).field === 'CustomerID') {
                    (columns as Column).visible = false;
                } else if ((columns as Column).field === 'ShipCity') {
                    (columns as Column).visible = true;
                }
            }
        }
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['Print'];
            this.pageOptions = { pageSize: 6 };
        }
    }
}

```

### 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 printing large data

Printing a large volume of data all at once in the grid can have certain limitations due to potential browser performance issues. Rendering numerous DOM elements on a single page can lead to browser slowdowns or even hang the browser. The grid offers a solution to manage extensive datasets through

virtualization. However, it's important to note that virtualization for both rows and columns is not feasible during the printing process.

If printing all the data remains a requirement, an alternative approach is recommended. Exporting the grid data to formats like [Excel](#) or [CSV](#) or [Pdf](#) is advised. This exported data can then be printed using non-web-based applications, mitigating the potential performance challenges associated with printing large datasets directly from the browser.

### Retain grid styles while printing

The Syncfusion Angular Grid provides a [beforePrint](#) event that allows you to customize the appearance and styles of the grid before it is sent to the printer. By handling this event, you can ensure that the grid retains its styles and appearance while printing.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems, PageSettingsModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
(beforePrint)='beforePrint()'
[allowPaging]='true' [pageSettings]='pageOptions'
[toolbar]='toolbarOptions'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`,
  styleUrls: ['app.component.css'],
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  public pageOptions?: PageSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
```



```

        this.toolbarOptions = ['Print'];
        this.pageOptions = { pageSize: 6 };
    }
    beforePrint() {
        var styleElement = document.createElement('style');
        styleElement.innerHTML = `
        .e-grid .e-headercell {
            background: #24a0ed !important;
        }
        .e-grid .e-row .e-rowcell {
            background: #cbefff !important;
        }
        .e-grid .e-altrow .e-rowcell{
            background: #e7d7f7 !important;
        }
        `;
        (document.querySelector('head') as
        Element).appendChild(styleElement);
    }
}

```

### MAIN.TS

```

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

```

### Print grid along with other components

To print the Syncfusion Angular Grid along with another component, such as a chart, you can use the [beforePrint](#) event of the grid. In this event, you can clone the content of the other component and append it to the print document.

Here is an example of how to print grid along with chart component:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ChartModule } from '@syncfusion/ej2-angular-charts'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { AccumulationChartModule } from '@syncfusion/ej2-angular-charts'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { PieSeriesService, AccumulationTooltipService,
AccumulationDataLabelService } from '@syncfusion/ej2-angular-charts'
import { LineSeriesService, DateTimeService,
DataLabelService, StackingColumnSeriesService, CategoryService, ChartShape,
StepAreaSeriesService, SplineSeriesService, ChartAnnotationService,
LegendService, TooltipService, StripLineService,
SelectionService, ScatterSeriesService
} from '@syncfusion/ej2-angular-charts'
import { Component, ElementRef, OnInit, ViewChild } from '@angular/core';
import { ChartComponent } from '@syncfusion/ej2-angular-charts';

```

```

import { GridComponent, ActionEventArgs, PageService, PrintEventArgs } from
 '@syncfusion/ej2-angular-grids';
import { Query, DataManager } from '@syncfusion/ej2-data';
import { orderData } from './datasource';
@Component({
  imports: [
    ChartModule, AccumulationChartModule, GridModule, ButtonModule
  ],
  providers: [ LineSeriesService, DateTimeService, DataLabelService,
    StackingColumnSeriesService, CategoryService,
    StepAreaSeriesService, SplineSeriesService,
    ChartAnnotationService, LegendService, TooltipService, StripLineService,
    PieSeriesService, AccumulationTooltipService,
    AccumulationDataLabelService, SelectionService, ScatterSeriesService
    , PageService],
  standalone: true,
  selector: 'app-root',
  template: `
    <button class ="printbtn" ejs-button cssClass="e-danger"
    (click)="onClick()"> Print </button>
    <ejs-grid #grid
      [dataSource]="data"
      [allowPaging]="true"
      [pageSettings]="pageSettings"
      printMode="CurrentPage"
      (dataBound)="dataBound()"
      (actionComplete)="actionComplete($event)"
      (beforePrint)="beforePrint($event)">
      <e-columns>
        <e-column field="OrderDate" headerText="Order Date"
width="130" format="yMd" textAlign="right"></e-column>
        <e-column field="Freight" width="120" format="C2"
textAlign="right"></e-column>
      </e-columns>
    </ejs-grid>
    <h4>Chart</h4>
    <div #chartContainer >
      <ejs-chart #chart id="chart-container"
        [primaryXAxis]="primaryXAxis"
        [primaryYAxis]="primaryYAxis"
        [title]="title">
        width="60%"
        <e-series-collection>
          <e-series type="Line" xName="OrderDate" yName="Freight"
width="1" columnWidth="0.4" name="dataview" [marker]="marker"></e-series>
        </e-series-collection>
      </ejs-chart>
    </div>
  `,
  providers: [PageService]
})
export class AppComponent implements OnInit {
  public primaryXAxis?: Object;
  public chartData?: Object[];
  public data?: Object[];
  public title?: string;
  public marker?: Object;

```

```

public primaryYAxis?: Object;
public pageSettings?: Object;
@ViewChild('chart') public chart?: ChartComponent;
@ViewChild('grid') public grid?: GridComponent;
@ViewChild('chartContainer') public chartContainer?: ElementRef;
ngOnInit(): void {
    this.data = new DataManager(orderData as JSON[]).executeLocal(new
Query().take(100));
    this.pageSettings = { pageSize: 10 };
}
dataBound() {
    this.chart!.primaryXAxis = {
        valueType: 'DateTime',
    };
    this.chart!.series[0].marker = { visible: true };
    this.chart!.series[0].xName = 'OrderDate';
    this.chart!.series[0].yName = 'Freight';
    this.chart!.series[0].dataSource = (this.grid as
GridComponent).getCurrentViewRecords();
}
onClick() {
    (this.grid as GridComponent).print();
}
beforePrint(args: PrintEventArgs) {
    if (this.chartContainer) {
        const clonedChartContainer =
this.chartContainer.nativeElement.cloneNode(true);
        (args.element as Element).appendChild(clonedChartContainer);
    }
    public actionComplete(args: ActionEventArgs): void {
        if (args.requestType === 'paging') {
            this.chart!.series[0].dataSource = (this.grid as
GridComponent).getCurrentViewRecords();
            this.chart?.refresh();
        }
    }
}
}

```

## MAIN.TS

```

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

```

## Hierarchy grid in Angular Grid component

The Hierarchy Grid in an Angular Grid component is typically used when you need to display hierarchical data in a tabular format with expandable and collapsible rows. It allows you to represent parent and child relationships within the grid, making it easier for you to navigate and understand the data.

This feature can be enabled by utilizing the [childGrid](#) and [childGrid.queryString](#) properties of the grid component.

To enable the Hierarchy Grid feature:

1. Inject the **DetailRowService** in the provider section of your **AppModule**. This service is essential for handling the hierarchy grid functionality.
2. Define the **childGrid** property within the Grid component configuration. This property describes the options of the child grid.
3. Specify the **childGrid.queryString** property to establish the relation between the parent and child grids and visualizes the data in a hierarchical structure. This property determines how the child records are fetched based on the parent record.

The following example demonstrates how to enable the hierarchy feature in the grid

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService,
    DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='parentData' height='315px'
[childGrid]='childGrid'>
    <e-columns>
      <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
      <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
      <e-column field='LastName' headerText='Last Name'
width=100></e-column>
      <e-column field='City' headerText='City'
width=120></e-column>
    </e-columns>
  </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public parentData?: object[];
  public childGrid: GridModel = {
    dataSource: data,
    queryString: 'EmployeeID',
    columns: [
      { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 90 },
      { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
```

```

        { field: 'ShipCity', headerText: 'Ship City', width: 100 },
        { field: 'ShipName', headerText: 'Ship Name', width: 120 }
    ],
    };
    ngOnInit(): void {
        this.parentData = employeeData;
    }
}

```

### MAIN.TS

```

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

```

\* Grid supports n level of child grids.

\* Hierarchical binding is not supported when [DetailTemplate](#) is enabled.

### Bind hierarchy grid with different field

By default, the parent and child grids have the same field name to map and render a hierarchical grid. However, the component supports establishing a parent-child relationship between grids with different field names. This feature is beneficial when you want to create a parent-child relationship between grids but need to use distinct field names for mapping the data. As a result, you can easily establish the desired relationship between the parent and child grids, even with different field names for data mapping.

By default, the parent and child grid relation is maintained using the [queryString](#) property, which requires the same field name for both grids. However, to achieve the parent and child relation with different fields, you need to modify the mapping value in the [load](#) event of child grid.

In the following example, the [load](#) event is utilized to customize the mapping value for the child grid. By accessing the `parentDetails` property and its `parentKeyFieldValue`, you can set the desired mapping value. The `parentRowData` property contains the data of the parent row, and by using the `EmployeeID` field name, you can extract the corresponding value from the parent row data.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService,
DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridModel, ParentDetails, GridComponent } from '@syncfusion/ej2-
angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [PageService,

```

```

        SortService,
        FilterService,
        GroupService, DetailRowService],
standalone: true,
selector: 'app-root',
template: `<ejs-grid #grid [dataSource]='parentData' height='265px'
[childGrid]='childGrid'>
    <e-columns>
        <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
        <e-column field='LastName' headerText='Last Name'
width=100></e-column>
        <e-column field='City' headerText='City'
width=150></e-column>
    </e-columns>
</ejs-grid>`,
    })
export class AppComponent implements OnInit {
    @ViewChild('grid')
    public grid?: GridComponent;
    public parentData?: object[];
    public childGrid: GridModel | GridComponent = {
        queryString: 'ID',
        dataSource: data,
        columns: [
            { field: 'OrderID', headerText: 'Order ID', width: 90 },
            { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
            { field: 'ShipCity', headerText: 'Ship City', width: 100 },
            { field: 'ShipName', headerText: 'Ship Name', width: 120 }
        ],
        load() {
            (this.parentDetails as ParentDetails).parentKeyFieldValue = (<{
EmployeeID?: string}>(this.parentDetails as
ParentDetails).parentRowData) ['EmployeeID'];
        }
    };
    ngOnInit(): void {
        this.parentData = employeeData;
    }
}

```

### MAIN.TS

```

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

```

Make sure to adjust the field name according to your specific scenario.

### Expand child grid initially

Expanding the child grid initially in the Syncfusion Angular Grid component is helpful when you want to display the child rows of the hierarchical grid expanded by default upon grid load. This can be beneficial

in scenarios where you want to provide immediate visibility into the hierarchical data without requiring you to manually expand each child row.

To achieve this, you can use the [expand](#) method with the desired target index (number) in the [dataBound](#) event of the grid.

In the provided example, expand the third record of the grid by utilizing the `expand` method within the `dataBound` event.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService,
    DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='parentData' height='265px'
[childGrid]='childGrid' (dataBound)='onDataBound()' '>
    <e-columns>
      <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
      <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
      <e-column field='LastName' headerText='Last Name'
width=100></e-column>
      <e-column field='City' headerText='City'
width=100></e-column>
    </e-columns>
  </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public parentData?: object[];
  public childGrid: GridModel = {
    dataSource: data,
    queryString: 'EmployeeID',
    columns: [
      { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 90 },
      { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
      { field: 'ShipCity', headerText: 'Ship City', width: 100 },
      { field: 'ShipName', headerText: 'Ship Name', width: 120 }
    ],
  },
}
```

```

};
@ViewChild('grid')
public grid?: GridComponent;
ngOnInit(): void {
    this.parentData = employeeData;
}
onDataBound(): void {
    (this.grid as GridComponent).detailRowModule.expand(2);
}
}

```

### MAIN.TS

```

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

```

Index values begin with “0”, allowing you to provide the desired target index to expand a specific child grid initially.

### Dynamically load child grid data

Dynamically load child grid data in Syncfusion Angular Grid helps improve performance, optimize data transmission, and enhance the your experience by providing on-demand access to relevant information. Additionally, it offers flexibility in data presentation, which helps improve the overall efficiency of your application.

To dynamically load the `dataSource` of a child grid in the Grid, you can utilize the `load` event of parent grid. This event allows you to customize the loading behavior of the child grid based on the data of parent grid.

The following example demonstrates how to dynamically load child grid data using the `load` event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService,
    DetailRowService],
  standalone: true,
  selector: 'app-root',

```



```

    template: `<ejs-grid #grid [dataSource]='parentData' height='265px'
    [childGrid]='childGrid' (load)='onLoad()'`>
        <e-columns>
            <e-column field='EmployeeID' headerText='Employee
            ID' textAlign='Right' width=80></e-column>
            <e-column field='FirstName' headerText='FirstName'
            width=100></e-column>
            <e-column field='LastName' headerText='Last Name'
            width=100></e-column>
            <e-column field='City' headerText='City'
            width=100></e-column>
        </e-columns>
    </ejs-grid>`,
  })
  export class AppComponent implements OnInit {
    public parentData?: object[];
    public childGrid: GridModel = {
      queryString: 'EmployeeID',
      columns: [
        { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
        width: 90 },
        { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
        { field: 'ShipCity', headerText: 'Ship City', width: 100 },
        { field: 'ShipName', headerText: 'Ship Name', width: 120 }
      ],
    };
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
      this.parentData = employeeData;
    }
    onLoad(): void {
      (this.grid as GridComponent).childGrid.dataSource = data; // assign
      data source for child grid.
    }
  }

```

## MAIN.TS

```

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

```

### Dynamically bind data to child grid based on parent row data

Dynamically binding data to a child grid based on the parent row data in the Syncfusion Angular Grid component is useful when you want to display child grid data that is specific to each parent row. This feature allows for a dynamic and contextual representation of data within the child grid.

To dynamically bind data to the child grid based on the parent row data instead of using the [queryString](#) property, you can utilize the [detailDataBound](#) event of the grid. This event is triggered when expanding the child grid.

In the `detailDataBound` event handler, you can filter the child grid's `dataSource` based on the `EmployeeID` column value of the parent row data. This can be achieved by using the `DataManager` plugin and applying a filter to the child grid's `dataSource`. The filtered data can be assigned as the new `dataSource` for the child grid. This can be demonstrated by the following sample.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { childData, employeeData, childColumnDataType } from './datasource';
import { DataManager, Query } from '@syncfusion/ej2-data';
import { GridModel, DetailDataBoundEventArgs, IGrid } from '@syncfusion/ej2-
angular-grids';
@Component({
imports: [

    GridModule
],
providers: [PageService,
SortService,
FilterService,
GroupService,
DetailRowService],
standalone: true,
selector: 'app-root',
template: `<ejs-grid [dataSource]='parentData' height='265px'
[childGrid]='childGrid' (detailDataBound)='detailDataBound($event)'>
    <e-columns>
        <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
        <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
        <e-column field='LastName' headerText='Last Name'
width=100></e-column>
        <e-column field='City' headerText='City'
width=100></e-column>
    </e-columns>
</ejs-grid>`,
})
export class AppComponent implements OnInit {
    public parentData?: object[];
    public childGrid: GridModel = {
        columns: [
            { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 90 },
            { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
            { field: 'ShipCity', headerText: 'Ship City', width: 100 },
            { field: 'ShipName', headerText: 'Ship Name', width: 120 }
        ],
    };
    ngOnInit(): void {
        this.parentData = employeeData;
    }
}
```

```

    }
    detailDataBound({data, childGrid} : DetailDataBoundEventArgs) {
        var empIdValue = (data as childColumnDataType).EmployeeID;
        var childGridData = new DataManager(childData).executeLocal(
            new Query().where('EmployeeID', 'equal', empIdValue, true)
        );
        (childGrid as IGrid).query = new Query();
        (childGrid as IGrid).dataSource = childGridData;
    }
}

```

## MAIN.TS

```

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

```

### Adding record in child grid

Adding a record in a child grid within the Syncfusion Angular Grid component is useful when you want to provide the ability to add new records to the child grid. This feature allows you to input and save additional data specific to each parent row.

To maintain the parent-child relationship in the Grid when adding a record to the child grid, you need to set the value for the `queryString` in the added data. This can be done using the [actionBegin](#) event.

In the following example, the parent and child grids are related by the **EmployeeID** field. To add a new record in the child grid, the **EmployeeID** field needs to be set with the value of the parent record's `queryString` in the `actionBegin` event.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
    GroupService, DetailRowService, EditService, ToolbarService } from
    '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data, employeeData, childColumnDataType } from './datasource';
import { AddEventArgs, GridModel, ParentDetails } from '@syncfusion/ej2-
angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [PageService,
        SortService,
        FilterService,
        GroupService, DetailRowService, EditService, ToolbarService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='parentData' height='265px'
[childGrid]='childGrid'>

```

```

        <e-columns>
            <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
            <e-column field='LastName' headerText='Last Name'
width=100></e-column>
            <e-column field='City' headerText='City'
width=100></e-column>
        </e-columns>
    </ejs-grid>`,
    })
    export class AppComponent implements OnInit {
        public parentData?: object[];
        public childGrid: GridModel = {
            dataSource: data,
            queryString: 'EmployeeID',
            toolbar: ['Add', 'Edit', 'Delete', 'Update', 'Cancel'],
            editSettings: { allowEditing: true, allowAdding: true,
allowDeleting: true },
            columns: [
                { field: 'OrderID', headerText: 'Order ID', isPrimaryKey: true,
textAlign: 'Right', width: 90 },
                { field: 'EmployeeID', headerText: 'Employee ID', textAlign:
'Right', allowEditing: false, width: 80 },
                { field: 'ShipCity', headerText: 'Ship City', width: 100 },
                { field: 'ShipName', headerText: 'Ship Name', width: 120 }
            ],
            actionBegin({ requestType, data }: AddEventArgs) {
                if (requestType === 'add') {
                    // `parentKeyFieldValue` refers to the queryString field
value of the parent record.
                    const parentFieldValue = (this.parentDetails as
ParentDetails)?.parentKeyFieldValue;
                    if (typeof parentFieldValue === 'number') {
                        (data as childColumnDataType).EmployeeID =
parentFieldValue;
                    }
                }
            }
        };
        ngOnInit(): void {
            this.parentData = employeeData;
        }
    }
}

```

### MAIN.TS

```

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

```

### Template column in child grid

A template column in a child grid within the Syncfusion Angular Grid component is valuable when you want to customize the appearance and functionality of specific columns in the child grid. It is useful for incorporating interactive elements, custom formatting, or complex data representation within specific columns of the child grid.

To achieve this, you can utilize the [template](#) property of a column to display a custom element instead of a field value in the Grid. Template columns defined in the child grid will be null in the **ngOnInit** method, which means they will not be shown in the UI. They will be rendered after the entire HTML view rendering process, and you can access and utilize them in the **ngAfterViewInit** method to display the template columns in the child grid.

During the [load](#) event of the child grid, it is necessary to set the 'registeredTemplate' to empty. This action will remove any previously existing templates. By doing so, you gain the flexibility to dynamically apply templates to the grid's cells based on different conditions or requirements.

The following example demonstrates, how to show a custom image in the **Employee Image** column of the child grid by utilizing the `template` property of the column.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild, ViewContainerRef, Inject,
AfterViewInit } from '@angular/core';
import { data, employeeData } from '../datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService, DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='parentData' height='315px'
[childGrid]='childGrid'>
    <e-columns>
      <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
      <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
      <e-column field='LastName' headerText='Last Name'
width=100></e-column>
      <e-column field='City' headerText='City'
width=100></e-column>
    </e-columns>
  </ejs-grid>
  <ng-template #childtemplate let-data>
    <div class="image">
```

```

        
        </div>
    </ng-template>`,
    })
    export class AppComponent implements OnInit, AfterViewInit {
        constructor(@Inject(ViewContainerRef) private viewContainerRef?:
ViewContainerRef) {
        }
        public parentData?: object[];
        @ViewChild('childtemplate', { static: true }) public childtemplate?:
any;
        public childGrid?: any;
        ngAfterViewInit() {
            this.childtemplate.elementRef.nativeElement._viewContainerRef =
this.viewContainerRef;
            this.childtemplate.elementRef.nativeElement.propName = 'template';
        }
        ngOnInit(): void {
            this.parentData = employeeData;
            this.childGrid = {
                dataSource: data,
                queryString: 'EmployeeID',
                load() {
                    this.registeredTemplate = {}; // set registerTemplate
value as empty in load event
                },
                columns: [
                    { headerText: 'Employee Image', textAlign: 'Center',
template: this.childtemplate, width: 120 },
                    { field: 'OrderID', headerText: 'Order ID', textAlign:
'Right', width: 90 },
                    { field: 'CustomerID', headerText: 'Customer ID', width: 100
},
                    { field: 'ShipCity', headerText: 'Ship City', width: 100 },
                    { field: 'ShipName', headerText: 'Ship Name', width: 120 }
                ],
            };
        }
    }
}

```

### 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 get parent detail in child grid

Getting parent details in a child grid in an Angular Grid component is useful when you want to display and utilize information from the parent row within the child grid. This can be beneficial in scenarios where you need to provide additional context or perform calculations based on the parent row's data

To achieve this, you can utilize the [created](#) event. This event is triggered when the child grid is created and can be used to handle the child grid.

The following example demonstrates how to obtain parent details in a child grid using the `created` event. Within the `created` event, you can access the parent row data using `this.parentDetails.parentRowData` and display the desired details in the message.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data, employeeData, ParentDetailsDataType } from './datasource';
import { GridModel, ParentDetails } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="margin-left:100px;">
      <p style="color:black; font-size: large;" id="message"></p>
    </div>
    <ejs-grid #grid [dataSource]='parentData' height='315px'
    [childGrid]='childGrid'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee ID'
        textAlign='Right' width=80></e-column>
        <e-column field='FirstName' headerText='FirstName'
        width=100></e-column>
        <e-column field='LastName' headerText='Last Name' width=100></e-
        column>
        <e-column field='City' headerText='City' width=120></e-column>
      </e-columns>
    </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public parentData?: object[];
  public message?: string;
  public parentDetails?: ParentDetails[];
  public childGrid: GridModel = {
    dataSource: data,
    queryString: 'EmployeeID',
    created: this.created,
    columns: [
      { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
        width: 90 },
      { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
```

```

        { field: 'ShipCity', headerText: 'Ship City', width: 100 },
        { field: 'ShipName', headerText: 'Ship Name', width: 120 }
    ],
    };
    created() {
        var parentRowData = (this.parentDetails as
ParentDetails).parentRowData; // 'this' refers to the instance of the child
grid.
        (document.getElementById('message') as HTMLElement).innerHTML =
`EmployeeID: ${ (parentRowData as ParentDetailsDataType).EmployeeID},
FirstName: ${ (parentRowData as ParentDetailsDataType).FirstName}, Title:
${ (parentRowData as ParentDetailsDataType).Title}`;
    }
    ngOnInit(): void {
        this.parentData = employeeData;
    }
}

```

### 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 by external button

The Hierarchy Grid in the Syncfusion Angular Grid component allows you to expand all child grid rows using an external button. This feature provides you with a convenient overview of all the hierarchical data within the grid, eliminating the need to manually expand each row individually.

By default, Grid renders all child grid rows in collapsed state. To expand all child grid rows in the Grid using an external button, you can utilize the [expandAll](#) method provided by the DetailRow module. Similarly, to collapse all grid rows, you can use the [collapseAll](#) method.

The following example demonstrates how to expand and collapse the hierarchy grid using an external button click function.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService,
GroupService, DetailRowService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridModel, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule,
        ButtonModule
    ],
    providers: [PageService,

```



```

        SortService,
        FilterService,
        GroupService,
        DetailRowService],
standalone: true,
selector: 'app-root',
template: `<button ejs-button (click)='expand()'>Expand All</button>
          <button ejs-button (click)='collapse()'>Collapse
All</button>
          <ejs-grid #grid [dataSource]='parentData' height='265px'
[childGrid]='childGrid'>
            <e-columns>
              <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right' width=80></e-column>
              <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
              <e-column field='LastName' headerText='Last Name'
width=100></e-column>
              <e-column field='City' headerText='City'
width=100></e-column>
            </e-columns>
          </ejs-grid>`,
}))
export class AppComponent implements OnInit {
  public parentData?: object[];
  public childGrid: GridModel = {
    dataSource: data,
    queryString: 'EmployeeID',
    columns: [
      { field: 'OrderID', headerText: 'Order ID', textAlign: 'Right',
width: 90 },
      { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
      { field: 'ShipCity', headerText: 'Ship City', width: 100 },
      { field: 'ShipName', headerText: 'Ship Name', width: 120 }
    ],
  };
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.parentData = employeeData;
  }
  expand(): void {
    (this.grid as GridComponent).detailRowModule.expandAll();
  }
  collapse(): void {
    (this.grid as GridComponent).detailRowModule.collapseAll();
  }
}

```

### MAIN.TS

```

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

```

The `expandAll` and `collapseAll` methods are not recommended for large datasets due to the considerable time it takes to update the changes in the UI.

#### Hide the expand/collapse icon in parent row when no record in child grid

The Syncfusion Angular Grid allows you to hide the expand/collapse icon in the parent row when there are no records in the child grid. However, in certain scenarios, you may want to hide the expand/collapse icon for parent rows that do not have any child records, providing a cleaner and more intuitive interface by eliminating unnecessary icons in empty parent rows.

To achieve this, you can utilize the [rowDataBound](#) event to hide the icon when there are no records in the child grid.

To hide the expand/collapse icon in parent row when no records in child grid, follow the given steps:

1. **Create a CSS Class with Custom Style:** Define a CSS class that overrides the default appearance of the Grid. This class will be used to customize the background color of the parent row when it is selected and when hovering over rows.

```
`css
.e-row[aria-selected="true"] .e-customizedexpandcell {
background-color: #e0e0e0;
}
.e-grid.e-gridhover tr[role='row']:hover {
background-color: #eee;
}
`
```

2. **Implement the rowDataBound Event Handler:** This event is triggered for each row in the grid when data is bound, allowing you to customize the row's appearance and behavior. In the provided code, the handler checks if the current row has any child records associated with it. If not, it hides the content of the first element, which contains the expand/collapse icon, and applies a custom CSS class (`e-customizedexpandcell`) to modify its appearance.

```
`typescript
public rowDataBound(args: RowDataBoundEventArgs) {
const parentData: number = (args.data as Employee)['EmployeeID'];
const childrecord: object[] = new DataManager(childData as JSON[]).
executeLocal(new Query().where('EmployeeID', 'equal', parentData, true));
if (childrecord.length === 0) {
// Here hide which parent row has no child records
const rowElement = args.row as HTMLTableRowElement;
const cellElement= rowElement.querySelector('td') as HTMLTableCellElement
```

```

cellElement.innerHTML = '';
cellElement.className = 'e-customizedexpandcell';
}
}
`

```

The following example demonstrates how to hide the expand/collapse icon in the row with **EmployeeID** as **1**, which does not have record in child Grid.

#### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService,
DetailRowService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { employeeData, childData } from './datasource';
import { RowDataBoundEventArgs, GridModel } from '@syncfusion/ej2-angular-
grids';
import { DataManager, Query } from '@syncfusion/ej2-data';
interface Employee {
    EmployeeID: number;
    FirstName: string;
    City: string;
    Country: string;
}
@Component({
    imports: [

        ButtonModule,
        GridModule
    ],
    providers: [PageService,
        SortService,
        FilterService,
        GroupService,
        DetailRowService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [childGrid]='childGrid'
(rowDataBound)="rowDataBound($event)">
        <e-columns>
            <e-column field='EmployeeID' headerText='EmployeeID'
width='80' ></e-column>
            <e-column field='FirstName' headerText='First Name'
width='100' ></e-column>
            <e-column field='City' headerText='City'
width='100'></e-column>
            <e-column field='Country' headerText='Country'
width='100'></e-column>
        </e-columns>
    </ejs-grid>`,

```

```

    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public childGrid?: GridModel;
        ngOnInit(): void {
            this.data = employeeData;
            this.childGrid = {
                dataSource: childData,
                queryString: 'EmployeeID',
                allowPaging: true,
                columns: [
                    { field: 'Order', headerText: 'Order ID', textAlign:
'Right', width: 120 },
                    { field: 'EmployeeID', headerText: 'Employee ID', textAlign:
'Right', width: 120 },
                    { field: 'ShipName', headerText: 'Ship Name', width: 150 }
                ]
            };
        }
        public rowDataBound(args: RowDataBoundEventArgs) {
            const parentData: number = (args.data as Employee)['EmployeeID'];
            const childrecord: object[] = new DataManager(childData as JSON[]).
                executeLocal(new Query().where('EmployeeID', 'equal',
parentData, true));
            if (childrecord.length === 0) {
                // Here hide which parent row has no child records
                const rowElement = args.row as HTMLTableRowElement;
                const cellElement = rowElement.querySelector('td') as
HTMLTableCellElement
                cellElement.innerHTML = ' ';
                cellElement.className = 'e-customizedexpandcell';
            }
        }
    }
}

```

### 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 child grid

The Syncfusion Angular Grid component offers various ways to customize the child grid appearance using both default CSS and custom themes. To access the child grid elements, you can use the **.e-detailcell** class selector, which targets the child grid.

#### Header

You can customize the appearance of the header elements in the child grid using CSS. Here are examples of how to customize the child grid header, header cell, and header cell div element.

#### Customizing the child grid header

To customize the appearance of the child grid header root element, you can use the following CSS code:

```

`css
.e-detailcell .e-grid .e-headercontent{
border: 2px solid green;
}
`

```

In this example, the `.e-detailcell` class targets the child grid and `.e-headercontent` targets its header root element. You can modify the `border` property to change the style of the header border. This customization allows you to override the thin line between the header and content of the child grid.

[illegible]

## Customizing the child grid header cell

To customize the appearance of the grid header cell elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-headercontent .e-headercell{
color: #ffffff;
background-color: #1ea8bd;
}
```

In this example, the `.e-headercell` class targets the header cell elements. You can modify the `color` and `background-color` properties to change the text color and background of the child grid's header cells.



### Customizing the child grid pager root element

To customize the appearance of the child grid pager root element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-gridpager {
font-family: cursive;
background-color: #deecf9;
}
`
```

In this example, the **.e-detailcell** class targets the child grid and the **.e-gridpager** class targets the pager root element. You can modify the **font-family** to change the font family and **background-color** property to change the background color of the pager.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY			
>	1	Nancy	Davolio	Seattle		
▼	2	Andrew	Fuller	Tacoma		
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME			
10250	HANAR	Rio de Janeiro	Hanari Carnes			
10252	SUPRD	Charleroi	Suprêmes délices			
10257	HILAA	San Cristóbal	HILARION-Abastos			
<<	<	1	2	>	>>	1 of 2 pages (6 items)

### Customizing the child grid pager container element

To customize the appearance of the child grid pager container element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-pagercontainer {
border: 2px solid #00b5ff;
font-family: cursive;
}
`
```

In this example, the **.e-pagercontainer** class targets the pager container element. You can modify the **border** property and **font-family** property to change the border color and font family of the pager container.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																									
>	1	Nancy	Davolio	Seattle																									
▼	2	Andrew	Fuller	Tacoma																									
<table> <tr> <th></th><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td></td><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr> <tr> <td></td><td>10252</td><td>SUPRD</td><td>Charleroi</td><td>Suprêmes délices</td></tr> <tr> <td></td><td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr> <tr> <td colspan="5"> <div> <span>&lt;&lt;</span> <span>&lt;</span> <span>1</span> <span>2</span> <span>&gt;</span> <span>&gt;&gt;</span> </div> <div>1 of 2 pages (6 items)</div> </td></tr> </table>						ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME		10250	HANAR	Rio de Janeiro	Hanari Carnes		10252	SUPRD	Charleroi	Suprêmes délices		10257	HILAA	San Cristóbal	HILARION-Abastos	<div> <span>&lt;&lt;</span> <span>&lt;</span> <span>1</span> <span>2</span> <span>&gt;</span> <span>&gt;&gt;</span> </div> <div>1 of 2 pages (6 items)</div>				
	ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																									
	10250	HANAR	Rio de Janeiro	Hanari Carnes																									
	10252	SUPRD	Charleroi	Suprêmes délices																									
	10257	HILAA	San Cristóbal	HILARION-Abastos																									
<div> <span>&lt;&lt;</span> <span>&lt;</span> <span>1</span> <span>2</span> <span>&gt;</span> <span>&gt;&gt;</span> </div> <div>1 of 2 pages (6 items)</div>																													

### Customizing the child grid pager navigation elements

To customize the appearance of the child grid pager navigation elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-gridpager .e-prevpagedisabled,
.e-detailcell .e-grid .e-gridpager .e-prevpage,
.e-detailcell .e-grid .e-gridpager .e-nextpage,
.e-detailcell .e-grid .e-gridpager .e-nextpagedisabled,
.e-detailcell .e-grid .e-gridpager .e-lastpagedisabled,
.e-detailcell .e-grid .e-gridpager .e-lastpage,
.e-detailcell .e-grid .e-gridpager .e-firstpage,
.e-detailcell .e-grid .e-gridpager .e-firstpagedisabled {
background-color: #deecf9;
}
```

In this example, the classes **.e-prevpagedisabled**, **.e-prevpage**, **.e-nextpage**, **.e-nextpagedisabled**, **.e-lastpagedisabled**, **.e-lastpage**, **.e-firstpage**, and **.e-firstpagedisabled** target the various pager navigation elements of the child grid. You can modify the **background-color** property to change the background color of these elements.



	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<div><table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>FREIGHT</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>65.83</td><td>Hanari Carnes</td></tr><tr><td>10252</td><td>SUPRD</td><td>51.3</td><td>Suprêmes délices</td></tr><tr><td>10253</td><td>HANAR</td><td>58.17</td><td>Hanari Carnes</td></tr></table><div><div>&lt;&lt;</div><div>&lt;</div><div>1</div><div>2</div><div>&gt;</div><div>&gt;&gt;</div></div><div>1 of 2 pages (6 items)</div></div>					ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME	10250	HANAR	65.83	Hanari Carnes	10252	SUPRD	51.3	Suprêmes délices	10253	HANAR	58.17	Hanari Carnes
ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME																	
10250	HANAR	65.83	Hanari Carnes																	
10252	SUPRD	51.3	Suprêmes délices																	
10253	HANAR	58.17	Hanari Carnes																	
>	3	Janet	Leverling	Kirkland																

## Customizing the child grid pager page numeric link elements

To customize the appearance of the child grid pager current page numeric link elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-gridpager .e-numericitem {
background-color: #5290cb;
color: #ffffff;
cursor: pointer;
}

.e-detailcell .e-grid .e-gridpager .e-numericitem:hover {
background-color: white;
color: #007bff;
}
`
```

In this example, the **.e-numericitem** class targets the page numeric link elements. You can modify the **background-color**, **color** properties to change the background color and text color of these elements.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy Davolio	Seattle																
▼	2	Andrew Fuller	Tacoma																
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10251</td><td>VICTE</td><td>Lyon</td><td>Victuailles en stock</td></tr><tr><td>10252</td><td>SUPRD</td><td>Charleroi</td><td>Suprêmes délices</td></tr></table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR	Rio de Janeiro	Hanari Carnes	10251	VICTE	Lyon	Victuailles en stock	10252	SUPRD	Charleroi	Suprêmes délices
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																
10250	HANAR	Rio de Janeiro	Hanari Carnes																
10251	VICTE	Lyon	Victuailles en stock																
10252	SUPRD	Charleroi	Suprêmes délices																
<div><div><div>&lt;&lt;</div><div>&lt;</div><div>1</div><div>2</div><div>3</div><div>4</div><div>&gt;</div><div>&gt;&gt;</div></div><div>1 of 4 pages (10 items)</div></div>																			

### Customizing the child grid pager current page numeric element

To customize the appearance of the child grid pager current page numeric element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-gridpager .e-currentitem {
background-color: #0078d7;
color: #fff;
}
```

In this example, the **.e-currentitem** class targets the current page numeric item. You can modify the **background-color** property to change the background color of this element and **color** property to change the text color.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																	
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10252</td><td>SUPRD</td><td>Charleroi</td><td>Suprêmes délices</td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR	Rio de Janeiro	Hanari Carnes	10252	SUPRD	Charleroi	Suprêmes délices	10257	HILAA	San Cristóbal	HILARION-Abastos
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																	
10250	HANAR	Rio de Janeiro	Hanari Carnes																	
10252	SUPRD	Charleroi	Suprêmes délices																	
10257	HILAA	San Cristóbal	HILARION-Abastos																	
<div><div><div>&lt;&lt;</div><div>&lt;</div><div>1</div><div>2</div><div>&gt;</div><div>&gt;&gt;</div></div><div>1 of 2 pages (6 items)</div></div>																				

### Sorting

You can customize the appearance of the sorting icons and multi sorting icons in the child grid using CSS. You can use the available Syncfusion [icons](#) based on your theme. Here's how to do it:

#### Customizing the child grid sorting icon

To customize the sorting icon that appears in the child grid header when sorting is applied, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-icon-ascending::before {
content: '\e7a3';
/ Icon code for ascending order /
}
.e-detailcell .e-grid .e-icon-descending::before {
content: '\e7b6';
/ Icon code for descending order /
}
`
```

In this example, the **.e-detailcell** class targets the child grid and the **.e-icon-ascending::before** class targets the sorting icon for ascending order, and the **.e-icon-descending::before** class targets the sorting icon for descending order.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																				
>	1	Nancy	Davolio	Seattle																				
▼	2	Andrew	Fuller	Tacoma																				
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>↓ ↑</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td></td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td></td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td></td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	CUSTOMER ID	↓ ↑	SHIP CITY	SHIP NAME	10250	HANAR		Rio de Janeiro	Hanari Carnes	10257	HILAA		San Cristóbal	HILARION-Abastos	10260	OTTIK		Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	↓ ↑	SHIP CITY	SHIP NAME																				
10250	HANAR		Rio de Janeiro	Hanari Carnes																				
10257	HILAA		San Cristóbal	HILARION-Abastos																				
10260	OTTIK		Köln	Ottilies Käseladen																				
>	3	Janet	Leverling	Kirkland																				

### Customizing the child grid multi sorting icon

To customize the multi sorting icon that appears in the child grid header when multiple columns are sorted, you can use the following CSS code:

`CSS

```
.e-detailcell .e-grid .e-sortnumber {
background-color: #deecf9;
font-family: cursive;
}
```

In this example, the **.e-sortnumber** class targets the background color and font family of the multi sorting icon. You can modify the **background-color** and **font-family** properties to customize the appearance of the multi sorting icon.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY
>	1	Nancy	Davolio	Seattle
▼	2	Andrew	Fuller	Tacoma
	ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME
	10250	HANAR	Rio de Janeiro	Hanari Carnes
	10257	HILAA	San Cristóbal	HILARION-Abastos
	10260	OTTIK	Köln	Ottilies Käseladen
>	3	Janet	Leverling	Kirkland

Filtering

You can customize the appearance of filtering elements in the child grid using CSS. Below are examples of how to customize various filtering elements, including filter bar cell elements, filter bar input elements, focus styles, clear icons, filter icons, filter dialog content, filter dialog footer, filter dialog input elements, filter dialog button elements, and Excel filter dialog number filters.

Customizing the child grid filter bar cell element

To customize the appearance of the filter bar cell element in the child grid header, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filterbar .e-filterbarcell {
background-color: #045fb4;
}
`
```

In this example, the **.e-detailcell** class targets the child grid and the **.e-filterbarcell** class targets the filter bar cell element in the child grid header. You can modify the **background-color** property to change the color of the filter bar cell element.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																					
>	1	Nancy	Davolio	Seattle																				
∨	2	Andrew	Fuller	Tacoma																				
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td><input type="text"/></td><td><input type="text"/></td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																					
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>																					
10250	HANAR	Rio de Janeiro	Hanari Carnes																					
10257	HILAA	San Cristóbal	HILARION-Abastos																					
10260	OTTIK	Köln	Ottilies Käseladen																					
>	3	Janet	Leverling	Kirkland																				

### Customizing the child grid filter bar input element

To customize the appearance of the filter bar input element in the child grid header, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filterbarcell .e-input-group input.e-input{
font-family: cursive;
}
```

In this example, the **.e-filterbarcell** class targets the filter bar cell element, and the **.e-input** class targets the input element within the cell. You can modify the **font-family** property to change the font of the filter bar input element.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																				
>	1	Nancy Davolio	Seattle																				
∨	2	Andrew Fuller	Tacoma																				
<table> <tr> <th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td></td><td>HILAA</td><td></td><td></td></tr> <tr> <td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr> <tr> <td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr> <tr> <td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr> </table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME		HILAA			10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																				
	HILAA																						
10250	HANAR	Rio de Janeiro	Hanari Carnes																				
10257	HILAA	San Cristóbal	HILARION-Abastos																				
10260	OTTIK	Köln	Ottilies Käseladen																				
>	3	Janet Leverling	Kirkland																				

### Customizing the child grid filter bar input focus

To customize the appearance of the child grid's filter bar input element's focus highlight, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filterbarcell .e-input-group.e-input-focus{
background-color: #deecf9;
}
`
```

In this example, the **.e-filterbarcell** class targets the filter bar cell element, and the **.e-input-group.e-input-focus** class targets the focused input element. You can modify the **background-color** property to change the color of the focus highlight.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																					
>	1	Nancy	Davolio	Seattle																				
▼	2	Andrew	Fuller	Tacoma																				
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td><input type="text"/></td><td><input type="text"/></td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																					
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>																					
10250	HANAR	Rio de Janeiro	Hanari Carnes																					
10257	HILAA	San Cristóbal	HILARION-Abastos																					
10260	OTTIK	Köln	Ottilies Käseladen																					
>	3	Janet	Leverling	Kirkland																				

### Customizing the child grid filter bar input clear icon

To customize the appearance of the child grid's filter bar input element's clear icon, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filterbarcell .e-input-group .e-clear-icon::before {
content: '\e72c';
}
`
```

In this example, the **.e-clear-icon** class targets the clear icon element within the input group. You can modify the **content** property to change the icon displayed.



EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																				
>	1	Nancy Davolio	Seattle																				
▼	2	Andrew Fuller	Tacoma																				
<table> <tr> <th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td></td><td>VIN</td><td></td><td></td></tr> <tr> <td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr> <tr> <td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr> <tr> <td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr> </table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME		VIN			10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																				
	VIN																						
10250	HANAR	Rio de Janeiro	Hanari Carnes																				
10257	HILAA	San Cristóbal	HILARION-Abastos																				
10260	OTTIK	Köln	Ottilies Käseladen																				
>	3	Janet Leverling	Kirkland																				

### Customizing the child grid child grid filtering icon

To customize the appearance of the filtering icon in the child grid header, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-icon-filter::before{
content: '\e81e';
}
```

In this example, the **.e-icon-filter** class targets the filtering icon element. You can modify the **content** property to change the icon displayed.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																	
10250	HANAR	Rio de Janeiro	Hanari Carnes																	
10257	HILAA	San Cristóbal	HILARION-Abastos																	
10260	OTTIK	Köln	Ottilies Käseladen																	
>	3	Janet	Leverling	Kirkland																

### Customizing the child grid filter dialog content

To customize the appearance of the child grid's filter dialog's content element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filter-popup .e-dlg-content {
background-color: #deecf9;
}
`
```

In this example, the **.e-filter-popup .e-dlg-content** classes target the content element within the filter dialog. You can modify the **background-color** property to change the color of the dialog's content.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy Davolio	Seattle																
▼	2	Andrew Fuller	Tacoma																
<table> <tr> <th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td>10250</td><td>HANAR</td><td></td><td></td></tr> <tr> <td>10257</td><td>HILAA</td><td></td><td>astos</td></tr> <tr> <td>10260</td><td>OTTIK</td><td></td><td>den</td></tr> </table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR			10257	HILAA		astos	10260	OTTIK		den
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																
10250	HANAR																		
10257	HILAA		astos																
10260	OTTIK		den																
>	3	Janet																	
>	4	Margaret																	
>	5	Steven																	
>	6	Michael																	
>	7	Robert																	
>	8	Laura																	
>	9	Anne																	

Sort A to Z  
Sort Z to A  
Clear Filter

Text Filters

☒ Select All  
☒ HANAR  
☒ HILAA  
☒ OTTIK

OK Cancel

### Customizing the child grid filter dialog footer

To customize the appearance of the child grid's filter dialog's footer element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filter-popup .e-footer-content {
background-color: #deecf9;
}
```

In this example, the **.e-filter-popup .e-footer-content** classes target the footer element within the filter dialog. You can modify the **background-color** property to change the color of the dialog's footer.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY	
>	1	Nancy	Davolio	Seattle
▼	2	Andrew	Fuller	Tacoma
<div><div><div>ORDER ID ▼</div><div>CUSTOMER ID ▼</div><div>SHIP CITY ▼</div><div>SHIP NAME ▼</div></div><div><div>10250</div><div>HANAR</div></div><div><div>10257</div><div>HILAA</div></div><div><div>10260</div><div>OTTIK</div></div></div>				
>	3	Janet		
>	4	Margaret		
>	5	Steven		
>	6	Michael		
>	7	Robert		
>	8	Laura		
>	9	Anne		

Sort A to Z

Sort Z to A

Clear Filter

Text Filters

Search

Q

☒ Select All

☒ HANAR

☒ HILAA

☒ OTTIK

OK

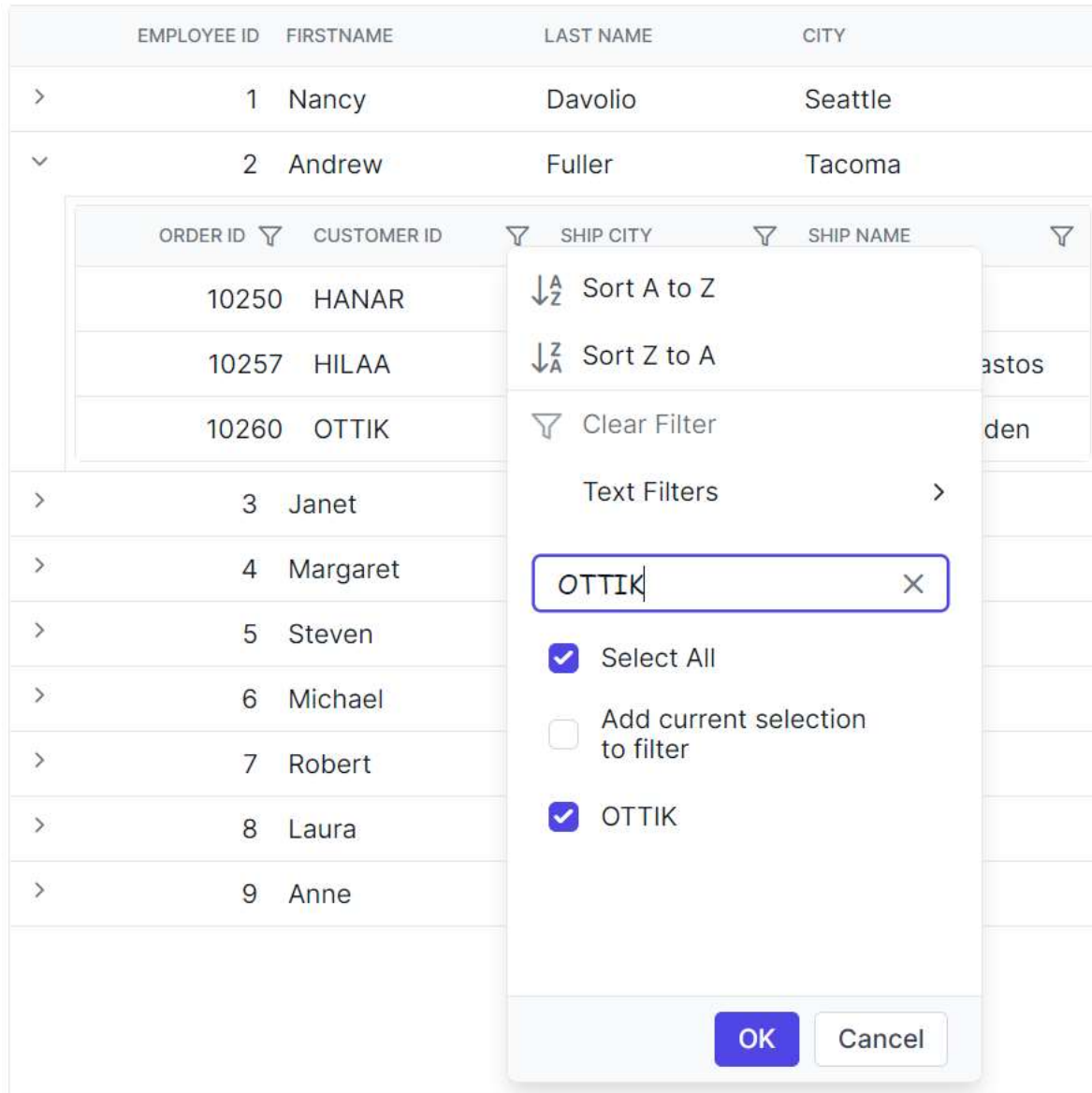
Cancel

### Customizing the child grid filter dialog input element

To customize the appearance of the child grid's filter dialog's input elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filter-popup .e-input-group input.e-input{
font-family: cursive;
}
```

In this example, the **.e-filter-popup** class targets the filter dialog, and the **.e-input** class targets the input elements within the dialog. You can modify the **font-family** property to change the font of the input elements.



### Customizing the child grid filter dialog button element

To customize the appearance of the child grid's filter dialog's button elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filter-popup .e-btn{
font-family: cursive;
}
```

In this example, the **.e-filter-popup** class targets the filter dialog, and the **.e-btn** class targets the button elements within the dialog. You can modify the **font-family** property to change the font of the button elements.

The screenshot displays a hierarchy grid with columns: EMPLOYEE ID, FIRSTNAME, LAST NAME, and CITY. The first row shows Employee ID 1, Nancy Davolio, Seattle. The second row shows Employee ID 2, Andrew Fuller, Tacoma. A child grid is expanded under Employee ID 2, showing columns: ORDER ID, CUSTOMER ID, SHIP CITY, and SHIP NAME. The child grid contains three rows: 10250 HANAR, 10257 HILAA, and 10260 OTTIK. An Excel filter dialog is open over the child grid, showing options to sort (A to Z, Z to A), clear filter, and text filters. The dialog also includes a search bar and a list of selected filters: Select All, HANAR, HILAA, and OTTIK. The dialog has OK and Cancel buttons at the bottom.

### Customizing the child grid excel filter dialog number filters element

To customize the appearance of the excel filter dialog's number filters in the child grid, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-filter-popup .e-contextmenu-wrapper ul{
background-color: #deecf9;
}
```

In this example, the **.e-filter-popup .e-contextmenu-wrapper** ul classes target the number filter elements within the excel filter dialog. You can modify the `background-color` property to change the color of these elements.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																	
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td></td><td></td></tr><tr><td>10257</td><td>HILAA</td><td></td><td>astos</td></tr><tr><td>10260</td><td>OTTIK</td><td></td><td>den</td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR			10257	HILAA		astos	10260	OTTIK		den
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																	
10250	HANAR																			
10257	HILAA		astos																	
10260	OTTIK		den																	
>	3	Janet																		
>	4	Margaret																		
>	5	Steven																		
>	6	Michael																		
>	7	Robert																		
>	8	Laura																		
>	9	Anne																		

Sort A to Z

Sort Z to A

Clear Filter

Text Filters

Search

☒ Select All

☒ HANAR

☒ HILAA

☒ OTTIK

OK

Cancel

### Grouping

You can customize the appearance of grouping elements in the child grid using CSS. Here are examples of how to customize the group header, group expand/collapse icons, group caption row, and grouping indent cell.

#### Customizing the child grid group header

To customize the appearance of the child grid's group header element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-groupdroparea {
background-color: #132f49;
}
```

In this example, the `.e-detailcell` class targets the child grid and the `.e-groupdroparea` class targets the group header element. You can modify the `background-color` property to change the color of the group header.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY
>	1	Nancy Davolio	Seattle
∨	2	Andrew Fuller	Tacoma
Drag a column header here to group its column			
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME
10250	HANAR	Rio de Janeiro	Hanari Carnes
10257	HILAA	San Cristóbal	HILARION-Abastos
10260	OTTIK	Köln	Ottilies Käseladen
>	3	Janet Leverling	Kirkland

Customizing the child grid group expand or collapse icons

To customize the appearance of the group expand/collapse icons in the child grid, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-icon-gdownarrow::before{
content:'\e7c9'
}
.e-detailcell .e-grid .e-icon-grightarrow::before{
content:'\e80f'
}
```

In this example, the `.e-icon-gdownarrow` and `.e-icon-grightarrow` classes target the expand and collapse icons, respectively. You can modify the `content` property to change the icon displayed. You can use the available Syncfusion icons based on your theme.



EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																					
>	1	Nancy Davolio	Seattle																					
▼	2	Andrew Fuller	Tacoma																					
<div>Customer ID ↑ ×</div> <table><tr><th>ORDER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td colspan="3">Customer ID: HANAR - 1 item</td></tr><tr><td colspan="3">Customer ID: HILAA - 1 item</td></tr><tr><td colspan="3">Customer ID: OTTIK - 1 item</td></tr><tr><td>10260</td><td>Köln</td><td>Ottilies Käseladen</td></tr><tr><td colspan="3">Customer ID: SUPRD - 1 item</td></tr><tr><td>10252</td><td>Charleroi</td><td>Suprêmes délices</td></tr></table>				ORDER ID	SHIP CITY	SHIP NAME	Customer ID: HANAR - 1 item			Customer ID: HILAA - 1 item			Customer ID: OTTIK - 1 item			10260	Köln	Ottilies Käseladen	Customer ID: SUPRD - 1 item			10252	Charleroi	Suprêmes délices
ORDER ID	SHIP CITY	SHIP NAME																						
Customer ID: HANAR - 1 item																								
Customer ID: HILAA - 1 item																								
Customer ID: OTTIK - 1 item																								
10260	Köln	Ottilies Käseladen																						
Customer ID: SUPRD - 1 item																								
10252	Charleroi	Suprêmes délices																						
>	3	Janet Leverling	Kirkland																					

### Customizing the child grid group caption row

To customize the appearance of the child grid's group caption row and the icons indicating record expansion or collapse, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-groupcaption {
background-color: #deecf9;
}
.e-detailcell .e-grid .e-recordplusexpand,
.e-detailcell .e-grid .e-recordpluscollapse {
background-color: #deecf9;
}
`
```

In this example, the **.e-groupcaption** class targets the group caption row element, and the **.e-recordplusexpand** and **.e-recordpluscollapse** classes target the icons indicating record expansion or collapse. You can modify the **background-color** property to change the color of these elements.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																						
>	1	Nancy	Davolio	Seattle																					
▼	2	Andrew	Fuller	Tacoma																					
<div>Customer ID ↑ ×</div> <table><tr><th>ORDER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td colspan="3">▼ Customer ID: HANAR - 1 item</td></tr><tr><td>10250</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td colspan="3">▼ Customer ID: HILAA - 1 item</td></tr><tr><td>10257</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td colspan="3">▼ Customer ID: OTTIK - 1 item</td></tr><tr><td>10260</td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	SHIP CITY	SHIP NAME	▼ Customer ID: HANAR - 1 item			10250	Rio de Janeiro	Hanari Carnes	▼ Customer ID: HILAA - 1 item			10257	San Cristóbal	HILARION-Abastos	▼ Customer ID: OTTIK - 1 item			10260	Köln	Ottilies Käseladen
ORDER ID	SHIP CITY	SHIP NAME																							
▼ Customer ID: HANAR - 1 item																									
10250	Rio de Janeiro	Hanari Carnes																							
▼ Customer ID: HILAA - 1 item																									
10257	San Cristóbal	HILARION-Abastos																							
▼ Customer ID: OTTIK - 1 item																									
10260	Köln	Ottilies Käseladen																							
>	3	Janet	Leverling	Kirkland																					

### Customizing the child grid grouping indent cell

To customize the appearance of the child grid's grouping indent cell element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-indentcell {
background-color: #deecf9;
}
```

In this example, the **.e-indentcell** class targets the grouping indent cell element. You can modify the **background-color** property to change the color of the indent cell.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																												
>	1	Nancy	Davolio	Seattle																												
▼	2	Andrew	Fuller	Tacoma																												
<div>Customer ID ↑ ×</div> <table><tr><th></th><th>ORDER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr><tr><td>▼</td><td colspan="3">Customer ID: HANAR - 1 item</td></tr><tr><td></td><td>10250</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr><tr><td>▼</td><td colspan="3">Customer ID: HILAA - 1 item</td></tr><tr><td></td><td>10257</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr><tr><td>▼</td><td colspan="3">Customer ID: OTTIK - 1 item</td></tr><tr><td></td><td>10260</td><td>Köln</td><td>Ottilies Käseladen</td></tr></table>						ORDER ID	SHIP CITY	SHIP NAME	▼	Customer ID: HANAR - 1 item				10250	Rio de Janeiro	Hanari Carnes	▼	Customer ID: HILAA - 1 item				10257	San Cristóbal	HILARION-Abastos	▼	Customer ID: OTTIK - 1 item				10260	Köln	Ottilies Käseladen
	ORDER ID	SHIP CITY	SHIP NAME																													
▼	Customer ID: HANAR - 1 item																															
	10250	Rio de Janeiro	Hanari Carnes																													
▼	Customer ID: HILAA - 1 item																															
	10257	San Cristóbal	HILARION-Abastos																													
▼	Customer ID: OTTIK - 1 item																															
	10260	Köln	Ottilies Käseladen																													
>	3	Janet	Leverling	Kirkland																												

### Toolbar

You can customize the appearance of the toolbar in the child grid using CSS. Here are examples of how to customize the toolbar root element and toolbar button element.

#### Customizing the child grid toolbar root element

To customize the appearance of the child grid's toolbar root element, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-toolbar-items {
background-color: #deecf9;
}
`
```

In this example, the **.e-detailcell** class targets the child grid and the **.e-toolbar-items** class targets the background color of the toolbar root element. You can modify the **background-color** property to change the background color of the toolbar.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy Davolio	Seattle																
∨	2	Andrew Fuller	Tacoma																
<div> <span>+ Add</span> <span>✎ Edit</span> <span>🗑 Delete</span> <span>💾 Update</span> <span>✕ Cancel</span> </div> <table> <tr> <th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr> <tr> <td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr> <tr> <td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr> </table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																
10250	HANAR	Rio de Janeiro	Hanari Carnes																
10257	HILAA	San Cristóbal	HILARION-Abastos																
10260	OTTIK	Köln	Ottilies Käseladen																
>	3	Janet Leverling	Kirkland																

### Customizing the child grid toolbar button element

To customize the appearance of the child grid's toolbar buttons, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-toolbar .e-btn {
background-color: #deecf9;
}
```

In this example, the **.e-toolbar .e-btn** selector targets the background color of the toolbar button elements. You can modify the **background-color** property to change the background color of the toolbar buttons.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																
>	1	Nancy Davolio	Seattle																
▼	2	Andrew Fuller	Tacoma																
<div> <span>+ Add</span> <span>✎ Edit</span> <span>🗑 Delete</span> <span>💾 Update</span> <span>✕ Cancel</span> </div> <table> <tr> <th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th></tr> <tr> <td>10250</td><td>HANAR</td><td>Rio de Janeiro</td><td>Hanari Carnes</td></tr> <tr> <td>10257</td><td>HILAA</td><td>San Cristóbal</td><td>HILARION-Abastos</td></tr> <tr> <td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käseladen</td></tr> </table>				ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	10250	HANAR	Rio de Janeiro	Hanari Carnes	10257	HILAA	San Cristóbal	HILARION-Abastos	10260	OTTIK	Köln	Ottilies Käseladen
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME																
10250	HANAR	Rio de Janeiro	Hanari Carnes																
10257	HILAA	San Cristóbal	HILARION-Abastos																
10260	OTTIK	Köln	Ottilies Käseladen																
>	3	Janet Leverling	Kirkland																

### Editing

You can customize the appearance of editing-related elements in the child grid using CSS. Below are examples of how to customize various editing-related elements.

#### Customizing the child grid edited and added row element

To customize the appearance of edited and added row table elements in the child grid, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-editedrow table,
.e-detailcell .e-grid .e-addedrow table {
background-color: #62b2eb;
}
```

In this example, the **.e-detailcell** class targets the child grid and the **.e-editedrow** class represents the edited row element, and the **.e-addedrow** class represents the added row element. You can modify the **background-color** property to change the color of these row table elements.

>

1

Nancy

Davolio

Seattle

▼


2


Andrew


Fuller

Tacoma

+ Add

 Edit

 Delete

 Update

×

 Cancel

ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME
10250	HANAR	Rio de Janeiro	Hanari Carnes
10257	HILAA	San Cristóbal	HILARION-Abastos
10260	OTTIK	Köln	Ottilies Käseladen

>

3

Janet

Leverling

Kirkland

EMPLOYEE ID

FIRSTNAME

LAST NAME

CITY

>

1

Nancy

Davolio

Seattle

▼


2


Andrew


Fuller

Tacoma

+ Add

 Edit

 Delete

 Update

×

 Cancel

ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME
10250	HANAR	Rio de Janeiro	Hanari Carnes
10257	HILAA	San Cristóbal	HILARION-Abastos
10260	OTTIK	Köln	Ottilies Käseladen

>

3

Janet

Leverling

Kirkland

Customizing the child grid edited row input element

To customize the appearance of edited row input elements in the child grid, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-editedrow .e-input-group input.e-input{
font-family: cursive;
```

```
color:rgb(214, 33, 123)
```

```
}
```

```
,
```

In this example, the **.e-editedrow** class represents the edited row element, and the **.e-input** class represents the input elements within the form. You can modify the **font-family** property to change the font and **color** property to change text color of the input elements.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY	
>	1	Nancy	Davolio	Seattle
∨	2	Andrew	Fuller	Tacoma
<div><div><div><div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div><div><div></div><div></div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div><div></div></div> <div><div></div>&lt;</div>				

### Customizing the child grid edit dialog header element

To customize the appearance of the edit dialog header element in the child grid, you can use the following CSS code:

```
`css
```

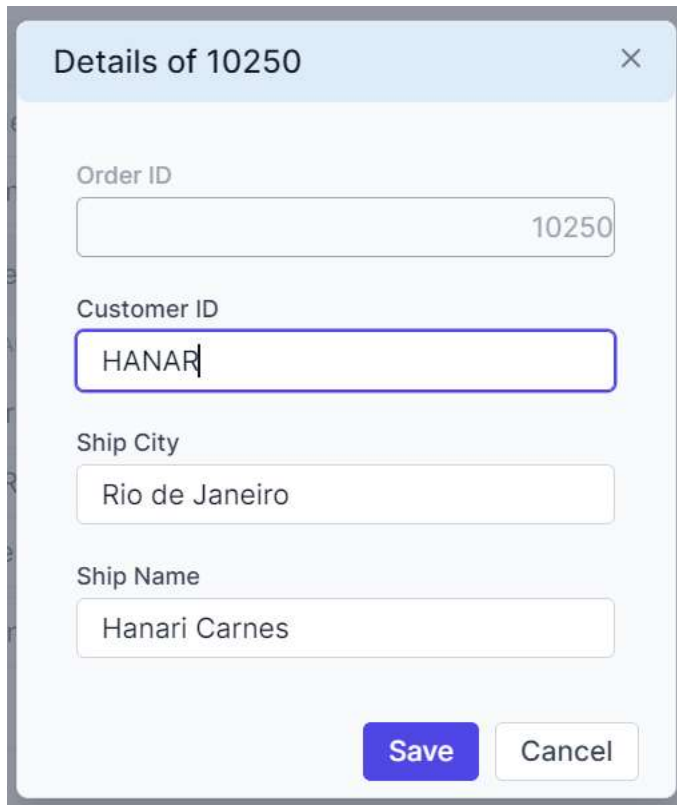
```
.e-edit-dialog .e-dlg-header-content {
```

```
background-color: #deecf9;
```

```
}
```

```
,
```

In this example, the **.e-edit-dialog** class represents the edit dialog, and the **.e-dlg-header-content** class targets the header content within the dialog. You can modify the **background-color** property to change the color of the header element.



Details of 10250

Order ID

10250

Customer ID

HANAR

Ship City

Rio de Janeiro

Ship Name

Hanari Carnes

Save Cancel

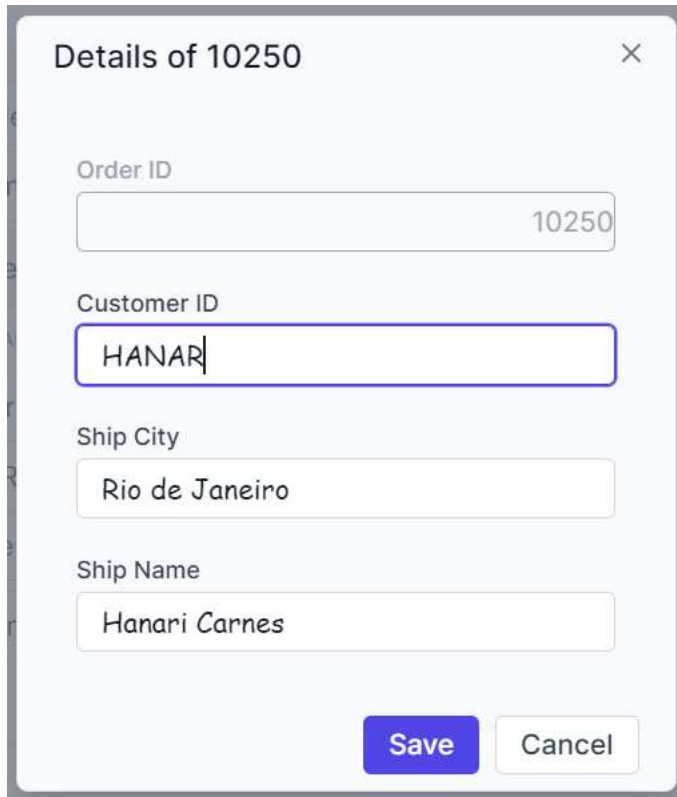
### Customizing the child grid edited row input element in dialog edit mode

To customize the appearance of the child grid's edited row input elements in dialog edit mode, you can use the following CSS code:

```
`css
.e-grid .e-gridform .e-rowcell .e-float-input .e-field {
font-family: cursive;
}
```

In this example, the **.e-gridform** class represents the editing form, and the **.e-float-input** class targets the floating input elements within the form. You can modify the **font-family** property to change the font of the input elements.





Details of 10250

Order ID  
10250

Customer ID  
HANAR

Ship City  
Rio de Janeiro

Ship Name  
Hanari Carnes

Save Cancel

### Customizing the child grid command column buttons

To customize the appearance of the child grid's command column buttons such as edit, delete, update, and cancel, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-delete::before, .e-grid .e-cancel-icon::before{
color: #f51717;
}
.e-detailcell .e-grid .e-edit::before, .e-grid .e-update::before {
color: #077005;
}
`
```

In this example, the **.e-edit**, **.e-delete**, **.e-update**, and **.e-cancel-icon** classes represent the respective command column buttons. You can modify the `color` property to change the color of these buttons.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																								
>	1	Nancy	Davolio	Seattle																								
▼	2	Andrew	Fuller	Tacoma																								
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th><th colspan="2">COMMANDS</th></tr><tr><td>10250</td><td>HANAR</td><td>Rio de Jan...</td><td>Hanari Carnes</td><td></td><td></td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristó...</td><td>HILARION-Ab...</td><td></td><td></td></tr><tr><td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käsel...</td><td></td><td></td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	COMMANDS		10250	HANAR	Rio de Jan...	Hanari Carnes			10257	HILAA	San Cristó...	HILARION-Ab...			10260	OTTIK	Köln	Ottilies Käsel...		
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	COMMANDS																								
10250	HANAR	Rio de Jan...	Hanari Carnes																									
10257	HILAA	San Cristó...	HILARION-Ab...																									
10260	OTTIK	Köln	Ottilies Käsel...																									
>	3	Janet	Leverling	Kirkland																								

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																								
>	1	Nancy	Davolio	Seattle																								
▼	2	Andrew	Fuller	Tacoma																								
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>SHIP CITY</th><th>SHIP NAME</th><th colspan="2">COMMANDS</th></tr><tr><td>10250</td><td>HANAR</td><td>Rio de J...</td><td>Hanari Carn...</td><td></td><td></td></tr><tr><td>10257</td><td>HILAA</td><td>San Cristó...</td><td>HILARION-Ab...</td><td></td><td></td></tr><tr><td>10260</td><td>OTTIK</td><td>Köln</td><td>Ottilies Käsel...</td><td></td><td></td></tr></table>					ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	COMMANDS		10250	HANAR	Rio de J...	Hanari Carn...			10257	HILAA	San Cristó...	HILARION-Ab...			10260	OTTIK	Köln	Ottilies Käsel...		
ORDER ID	CUSTOMER ID	SHIP CITY	SHIP NAME	COMMANDS																								
10250	HANAR	Rio de J...	Hanari Carn...																									
10257	HILAA	San Cristó...	HILARION-Ab...																									
10260	OTTIK	Köln	Ottilies Käsel...																									
>	3	Janet	Leverling	Kirkland																								

### Aggregate

You can customize the appearance of aggregate elements in the child grid using CSS. Below are examples of how to customize the aggregate root element and the aggregate cell elements.

#### Customizing the child grid aggregate root element

To customize the appearance of the child grid's aggregate root elements, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-gridfooter {
font-family: cursive;
}
```

In this example, the **.e-detailcell** class targets the child grid and the **e-gridfooter** class represents the root element of the aggregate row in the grid footer. You can modify the **font-family** property to change the font of the aggregate root element.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																								
>	1	Nancy	Davolio	Seattle																								
▼	2	Andrew	Fuller	Tacoma																								
	<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>FREIGHT</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>65.83</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>81.91</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>55.09</td><td>Ottilies Käseladen</td></tr><tr><td colspan="3">Sum: \$202.83</td><td></td></tr><tr><td colspan="3">Max: \$81.91</td><td></td></tr></table>				ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME	10250	HANAR	65.83	Hanari Carnes	10257	HILAA	81.91	HILARION-Abastos	10260	OTTIK	55.09	Ottilies Käseladen	Sum: \$202.83				Max: \$81.91			
	ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME																								
	10250	HANAR	65.83	Hanari Carnes																								
	10257	HILAA	81.91	HILARION-Abastos																								
	10260	OTTIK	55.09	Ottilies Käseladen																								
	Sum: \$202.83																											
	Max: \$81.91																											
>	3	Janet	Leverling	Kirkland																								

### Customizing the child grid aggregate cell elements

To customize the appearance of the child grid's aggregate cell elements (summary row cell elements), you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-summaryrow .e-summarycell {
background-color: #deecf9;
}
`
```

In this example, the **e-summaryrow** class represents the summary row containing aggregate cells, and the **e-summarycell** class targets individual aggregate cells within the summary row. You can modify the **background-color** property to change the **color** of the aggregate cell elements.



EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																	
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<div><table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>FREIGHT</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>65.83</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>81.91</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>55.09</td><td>Ottilies Käseladen</td></tr></table></div>					ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME	10250	HANAR	65.83	Hanari Carnes	10257	HILAA	81.91	HILARION-Abastos	10260	OTTIK	55.09	Ottilies Käseladen
ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME																	
10250	HANAR	65.83	Hanari Carnes																	
10257	HILAA	81.91	HILARION-Abastos																	
10260	OTTIK	55.09	Ottilies Käseladen																	
>	3	Janet	Leverling	Kirkland																

## Customizing the child grid cell selection background

To customize the appearance of the child grid's cell selection, you can use the following CSS code:

`CSS

```
.e-detailcell .e-grid td.e-cellselectionbackground {
```

```
background-color: #00b7ea;
```

}

•

In this example, the `.e-cellselectionbackground` class targets the background color of the cell selection. You can modify the `background-color` property to change the background color of the selected cells.

EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY																	
>	1	Nancy	Davolio	Seattle																
▼	2	Andrew	Fuller	Tacoma																
<table><tr><th>ORDER ID</th><th>CUSTOMER ID</th><th>FREIGHT</th><th>SHIP NAME</th></tr><tr><td>10250</td><td>HANAR</td><td>65.83</td><td>Hanari Carnes</td></tr><tr><td>10257</td><td>HILAA</td><td>81.91</td><td>HILARION-Abastos</td></tr><tr><td>10260</td><td>OTTIK</td><td>55.09</td><td>Ottilies Käseladen</td></tr></table>					ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME	10250	HANAR	65.83	Hanari Carnes	10257	HILAA	81.91	HILARION-Abastos	10260	OTTIK	55.09	Ottilies Käseladen
ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME																	
10250	HANAR	65.83	Hanari Carnes																	
10257	HILAA	81.91	HILARION-Abastos																	
10260	OTTIK	55.09	Ottilies Käseladen																	
>	3	Janet	Leverling	Kirkland																

### Customizing the child grid column selection background

To customize the appearance of the child grid's column selection, you can use the following CSS code:

```
`css
.e-detailcell .e-grid .e-columnselection {
background-color: #aec2ec;
}
`
```

In this example, the **.e-columnselection** class targets the background color of the column selection. You can modify the `background-color` property to change the background color of the selected columns.

	EMPLOYEE ID	FIRSTNAME	LAST NAME	CITY
>	1	Nancy	Davolio	Seattle
▼	2	Andrew	Fuller	Tacoma
	ORDER ID	CUSTOMER ID	FREIGHT	SHIP NAME
	10250	HANAR	65.83	Hanari Carnes
	10257	HILAA	81.91	HILARION-Abastos
	10260	OTTIK	55.09	Ottilies Käseladen
>	3	Janet	Leverling	Kirkland

See Also

- [Multiple querystring in hierarchy child grid in Angular Grid](#)

### State Management in Angular Grid component

State management in the Angular Grid component allows you to maintain the grid's state even after a browser refresh or when navigating to a different page within the same browser session. This feature is particularly useful for retaining the grid's configuration and data even after a page reload.

To enable state persistence in the Grid, you can utilize the [enablePersistence](#) property. When this property is set to **true**, the grid will automatically save its state in the browser's [localStorage](#), ensuring that the state is preserved across page reloads.

```
`html
<ejs-grid [dataSource]="data" [enablePersistence]="true"></ejs-grid>
`
```

The grid will store the state using the combination of the component name and component ID in the storage. For example, if the component name is **grid** and the ID is **OrderDetails**, the state will be stored as **gridOrderDetails**.

When enabling state persistence, the following grid settings will persist in the local storage.

| Grid Settings | Properties

persist

| Ignored properties

|

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

-----

----- | -----

-----

-----

----- |

| pageSettings | currentPage<br> pageCount<br> pageSize<br> pageSizes<br>

totalRecordsCount

| template<br>

enableQueryString

|

| groupSettings | allowReordering<br> columns<br> disablePageWiseAggregates<br>

enableLazyLoading<br> showDropArea<br> showGroupedColumn<br> showToggleButton<br>

showUngroupButton

|

captionTemplate

|

| columns | allowEditing<br> allowFiltering<br> allowGrouping<br> allowReordering<br>

allowResizing<br> allowSearching<br> allowSorting<br> autoFit<br> disableHtmlEncode<br>

enableGroupByFormat<br> field<br> foreignKeyField<br> index<br> showColumnMenu<br>

showInColumnChooser<br> sortDirection<br> type<br> uid<br> visible<br> width | clipMode<br>

commands<br> customAttributes<br> dataSource<br> defaultValue<br> displayAsCheckBox<br>

edit<br> editTemplate<br> editType<br> filter<br> filterBarTemplate<br> filterTemplate<br>

foreignKeyValue<br> format<br> formatter<br> freeze<br> headerTemplate<br> headerText<br>

headerTextAlign<br> headerValueAccessor<br> hideAtMedia<br> isFrozen<br> isIdentity<br>

isPrimaryKey<br> lockColumn<br> maxWidth<br> minWidth<br> sortComparer<br> template<br>

textAlign<br> validationRules<br> valueAccessor |

| sortSettings | -

| -

|

| filterSettings | -

|

-

|

| searchSettings | -

|

-

|

```
| selectedRowIndex | -
| -
|
```

The grid will persist only the last selected row index.

### Restore initial Grid state

In the Syncfusion Angular Grid component, you have the capability to restore the grid to its initial state, reverting all changes and configurations made during the interaction. This feature can be particularly useful when you want to reset the grid to its original settings, eliminating any applied filters, sorting, or column reordering.

Here are the steps to reset the grid to its initial state, even when the [enablePersistence](#) property is enabled:

### Changing component id

If you want to restore the initial state of the grid, consider changing the component ID. This step ensures that the grid is treated as a new instance, effectively reverting to its default settings.

Here is an example code to change the component id dynamically to restore initial grid state.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button id='restore'
(click)='clickHandler()'>Restore to initial state</button>
    <ej2-grid #grid [id]='gridId' [dataSource]='data'
[enablePersistence]='true' [allowPaging]='true' [allowFiltering]='true'
height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ej2-grid>`,
  providers: [GroupService, FilterService, PageService]
})
```



```
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public gridId?: string = 'OrderDetails'; // id for the Grid component
  ngOnInit(): void {
    this.data = data;
  }
  clickHandler() {
    this.gridId = `OrderDetails` + Math.floor(Math.random() * 10);
    location.reload();
  }
}
```

## MAIN.TS

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

### Clearing local storage

Another method to reset the grid is by clearing the local storage associated with the grid component. This action removes any stored state information, allowing the grid to return to its original configuration.

Here is an example code on how to clear local storage to retain its default state.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent } from '@syncfusion/ej2-
angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej2-button id='restore' (click)='clickHandler()'
  cssClass='e-primary'>Restore to initial state</button>
    <ej2-grid #grid id="OrderDetails" [dataSource]='data'
  [enablePersistence]='true' [allowPaging]='true' [allowFiltering]='true'
    height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
  textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
  width=150></e-column>
```

```

        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
</ejs-grid>,
    providers: [GroupService, FilterService, PageService]
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    clickHandler() {
        const grid = this.grid as GridComponent;
        grid.enablePersistence = false;
        window.localStorage.setItem("gridOrderDetails", ""); // Here grid is
component name and OrderDetails is component ID
        grid.destroy();
        location.reload();
    }
}

```

### MAIN.TS

```

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

```

### Restore to specific state version

Syncfusion Angular Grid supports version-based persistence for easy restoration to a specific state. To enable version based persistence, import `enableVersionBasedPersistence` from `@syncfusion/ej2-base` and set it globally to `true`. Define the grid in the template with properties, bind data, and configure persistence using [enablePersistence](#) and [ej2state-persistenceVersion](#).

In the below example, the `clickHandler` method is responsible for handling button clicks corresponding to different versions. Inside this method, the targeted version is assigned to the grid's `ej2state-persistenceVersion` dynamically. The code checks if there is already a persisted state for the selected version in the local storage. If found, the grid is updated with the settings retrieved from the local storage, including columns, filter settings, group settings, sort settings, page settings, and selected row index. If no persisted state is found, the current grid state is persisted to the local storage using the [getPersistData](#) method.

Here is an example of how to integrate version-based persistence into your Angular component and restore to specific state version:

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'

```

```

import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent, GroupService,
SortService, ReorderService } from '@syncfusion/ej2-angular-grids';
import { enableVersionBasedPersistence } from '@syncfusion/ej2-base';
enableVersionBasedPersistence(true);
@Component({
imports: [

    GridModule,
    ButtonModule
],
standalone: true,
selector: 'app-root',
template: `<h4 id='message'>{{message}}</h4>
<button ejs-button *ngFor="let v of versions" [id]='restore' + v"
(click)="clickHandler('v.' + v)">Version {{ v }}</button>
    <ejs-grid #grid id="OrderDetails" [dataSource]='data'
[enablePersistence]='true' [ej2StatePersistenceVersion]='gridversion'
[allowPaging]='true' [allowFiltering]='true'
[allowReordering]='true' [allowSorting]='true' [allowGrouping]='true'
height='210px'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
        </e-columns>
    </ejs-grid>`,
providers: [FilterService, PageService, GroupService, SortService,
ReorderService]
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
    public grid?: GridComponent;
    public message?: string;
    public gridversion?: string = 'v.0';
    public versions: number[] = [1, 2, 3];
    ngOnInit(): void {
        this.data = data;
    }
    clickHandler(version: string) {
        const grid = this.grid as GridComponent;
        grid.ej2StatePersistenceVersion = version;
        var persistedGridSettings: string =
(window.localStorage.getItem(`gridOrderDetails` +
grid.ej2StatePersistenceVersion)) as string;
        if (persistedGridSettings) {
            grid.setProperties(JSON.parse(persistedGridSettings));
            this.message = `Grid state restored to ` + version;
        } else {

```

```

        var gridData = grid.getPersistData();
        window.localStorage.setItem(`gridOrderDetails` +
grid.ej2StatePersistenceVersion), gridData);
    }
}
}

```

### MAIN.TS

```

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

```

### Restore to previous state

The Syncfusion Angular Grid component allows you to save and restore its state using local storage. This feature is helpful when you want to preserve the current state of the Grid, such as column order, sorting, and filtering, so that you can return to your previous work or configurations.

To implement this functionality, use the `getItem` and `setItem` methods for local storage, along with the Grid component's `setProperties` and `getPersistData` methods.

The provided code demonstrates how to save and restore the previous state of a Syncfusion Angular Grid component using local storage.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, FilterService, ToolbarService,
SortService, EditService, } from '@syncfusion/ej2-angular-grids'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import {
    GroupService,
    SortService,
    GridComponent,
    EditService,
    ToolbarService,
} from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonAllModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'app.component.html',
  providers: [GroupService, ToolbarService, SortService, EditService,
FilterService],
})
export class AppComponent {
  public data?: Object[];

```

```

public groupOptions?: Object;
public pageSettings?: Object;
public editSettings?: Object;
public filterOptions?: Object;
public toolbar?: string[];
public orderidrules?: Object;
public customeridrules?: Object;
public freightrules?: Object;
public editparams?: Object;
public formatoptions?: Object;
@ViewChild("Orders")
public grid?: GridComponent;
public state?: GridComponent;
public message?: string;
ngOnInit(): void {
    this.data = data;
    this.groupOptions = { showGroupedColumn: false, columns: ["ShipCountry"]
};
    this.filterOptions = { type: "Menu" };
    this.pageSettings = { pageCount: 5 };
    this.editSettings = { allowEditing: true };
    this.toolbar = ["Edit", "Update", "Cancel"];
    this.orderidrules = { required: true, number: true };
    this.customeridrules = { required: true };
    this.freightrules = { required: true };
    this.editparams = { params: { popupHeight: "300px" } };
    this.formatoptions = { type: "dateTime", format: "M/d/y hh:mm a" };
}
actionBegin() {
    this.message = "";
}
// Save grid state to local storage
save() {
    var persistData = (this.grid as GridComponent).getPersistData(); // Grid
persistData
    window.localStorage.setItem("gridOrders", persistData);
    this.message = "Grid state saved.";
}
// Restore grid state from local storage
restore() {
    let value: string = window.localStorage.getItem("gridOrders") as string;
    // "gridOrders" is component name + component id.
    this.state = JSON.parse(value);
    if (this.state) {
        (this.grid as GridComponent).setProperties(this.state);
        this.message = "Previous grid state restored.";
    } else {
        this.message = "No saved state found.";
    }
}
}

```

### APP.COMPONENT.HTML

```

<div class="control-section">
    <button ej-button class="e-success" (click)="save()">Save</button>

```

```

<button ej-button class="e-danger" (click)="restore()">restore</button>
<div id='message'>{{message}}</div>
<ejs-grid #Orders id="Orders" [dataSource]="data" allowPaging="true"
allowSorting="true" allowFiltering="true"
    [allowGrouping]="true" [editSettings]="editSettings"
[groupSettings]="groupOptions"
    [filterSettings]="filterOptions" [toolbar]="toolbar"
[pageSettings]="pageSettings" [enablePersistence]="true"
    height="320" (actionBegin)="actionBegin()">
    <e-columns>
        <e-column field="OrderID" headerText="Order ID" width="140"
textAlign="Right" isPrimaryKey="true"
            [validationRules]="orderidrules"></e-column>
        <e-column field="CustomerID" headerText="Customer ID"
width="140" [validationRules]="customeridrules">
        </e-column>
        <e-column field="Freight" headerText="Freight" width="140"
format="C2" textAlign="Right"
            editType="numericedit" [validationRules]="freightrules"></e-
column>
        <e-column field="OrderDate" [allowGrouping]="false"
headerText="Order Date" width="120"
            editType="datetimepickeredit" [format]="formatoptions"
textAlign="Right"></e-column>
        <e-column field="ShipCountry" headerText="Ship Country"
width="150" editType="dropdownedit"
            [edit]="editparams"></e-column>
    </e-columns>
</ejs-grid>
</div>

```

## MAIN.TS

```

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

```

## Maintaining custom query in a persistent state

When [enablePersistence](#) is enabled, the Grid does not automatically maintain custom query parameters after a page load. This is because the Grid refreshes its query params for every page load. You can maintain the custom query params by resetting the [addParams](#) method in the [actionBegin](#) event.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FilterService, GridModule, GroupService } from '@syncfusion/ej2-
angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent } from '@syncfusion/ej2-
angular-grids';
@Component({
imports: [

```

```

        GridModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #Orders [dataSource]='data'
[enablePersistence]='true' [allowPaging]='true' [allowFiltering]='true'
        height='210px' (actionBegin)='actionBegin()' '>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
                <e-column field='EmployeeID' headerText='Employee ID'
width=150></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
            </e-columns>
        </ejs-grid>`,
    providers: [GroupService, FilterService, PageService]
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('Orders')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    actionBegin() {
        (this.grid as GridComponent).query.addParams('dataSource', 'data');
    }
}

```

### MAIN.TS

```

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

```

### Get or set local storage value

If the [enablePersistence](#) property is set to **true**, the Grid property value is saved in the **window.localStorage** for reference. You can get or set the localStorage value by using the **getItem** and **setItem** methods in **window.localStorage**.

To retrieve the Grid model from Local Storage, follow these steps:

```
`typescript
```

```
//get the Grid model.
```

```
let value: string = window.localStorage.getItem('gridOrders'); // "gridOrders" is component name +
component id.
```

```
let model: Object = JSON.parse(value);
```

```
,
```

```
`typescript
```

```
//set the Grid model.
```

```
window.localStorage.setItem('gridOrders', JSON.stringify(value)); //"gridOrders" is component name +
component id.
```

```
,
```

### Prevent columns from persisting

In the Syncfusion Angular Grid component, you may sometimes want to prevent certain settings from being persisted when using the [enablePersistence](#) feature. When the `enablePersistence` property is set to `true`, the Grid properties such as [Grouping](#), [Paging](#), [Filtering](#), [Sorting](#), and [Columns](#) will persist. You can use the `addOnPersist` method to prevent these Grid properties from persisting.

The following example demonstrates how to prevent Grid columns from persisting. In the [dataBound](#) event of the Grid, you can override the `addOnPersist` method and remove the columns from the key list given for persistence.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent, Column } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button id='add' (click)='addColumn()'>Add
Columns</button>
      <button ejs-button id='remove'
(click)='removeColumn()'>Remove Columns</button>
      <ejs-grid #grid id="Orders" [dataSource]='data'
[enablePersistence]='true' [allowPaging]='true' height='210px'
(dataBound)='dataBound()'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
          <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
          <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
```



```

        </e-columns>
    </ejs-grid>`,
    providers: [GroupService, FilterService, PageService]
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
    }
    dataBound() {
        let cloned = (this.grid as any).addOnPersist;
        (this.grid as any).addOnPersist = function (key: any) {
            key = key.filter((item: string) => item !== "columns");
            return cloned.call(this, key);
        };
    }
    addColumn() {
        let obj = { field: "Freight", headerText: 'Freight', width: 120 };
        ((this.grid as GridComponent).columns as Column[]).push(obj as
Column); //you can add the columns by using the Grid columns method
        (this.grid as GridComponent).refreshColumns();
    }
    removeColumn() {
        (this.grid as GridComponent).columns.pop();
        (this.grid as GridComponent).refreshColumns();
    }
}

```

## MAIN.TS

```

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

```

### Add to persist

Persistence can be added to a Syncfusion Angular Grid component to enhance the user experience. Persistence allows saving and restoring the state of the grid, including column layouts, sorting, filtering, and other user-specific settings. In this documentation, you will explore how to persist column templates, header templates, and header text settings in the Angular Grid.

### Add a new column in persisted columns list

When the [enablePersistence](#) property is set to true in the Syncfusion Grid component, column configurations are persisted. If you need to add new columns to the existing persisted state, you can achieve this by using the Grid's built-in methods like `column.push`, and then call the [refreshColumns](#) method to update the UI with the new columns.

Here's an example of how to add a new column to a list of persisted columns:

## APP.COMPONENT.TS

```

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

```

```

import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent, Column } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ejs-button id='add' (click)='addColumn()'>Add
Columns</button>
      <ejs-grid #grid id="Orders" [dataSource]='data'
[enablePersistence]='true' [allowPaging]='true' height='210px'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
          <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
          <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
        </e-columns>
      </ejs-grid>`,
  providers: [GroupService, FilterService, PageService]
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  addColumn() {
    let obj = { field: "Freight", headerText: 'Freight', width: 120 };
    ((this.grid as GridComponent).columns as Column[]).push(obj as
Column); //you can add the columns by using the Grid columns method
    (this.grid as GridComponent).refreshColumns();
  }
}

```

## MAIN.TS

```

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

```

\* Adding new columns using `ColumnDirectives` directly in the grid initialization is not recommended if you intend to persist the new columns with the existing columns list

### *Persist the column template, header template and header text*

By default, when the [enablePersistence](#) property is set to **true** in the Syncfusion Grid component, certain column properties such as column template, header text, header template, column formatter, and value accessor are not persisted. This is because these properties can be customized at the application level.

To restore these column properties and achieve persistence, you can follow the approach of cloning the grid's columns property using JavaScript Object's assign method and manually storing it along with the persist data. When restoring the settings, this cloned column object must be assigned to the grid's columns property to restore the column settings. The following sample demonstrates this process:

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { ButtonAllModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { FilterService, PageService, GridComponent, Column } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id='message'>{{message}}</div><button ej2-button
id='save' (click)='save()'>Save column settings</button><button ej2-button
id='restore' (click)='restore()'>Restore column settings</button>
    <ejs-grid #Grid id="Orders" [dataSource]='data'
[enablePersistence]='true' [allowPaging]='true' [allowFiltering]='true'
height='210px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120>
        </e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150 headerTemplate='<button ej2-button>HeaderTemplate</button>'>
        </e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150 template='#template'>
        </e-column>
      </e-columns>
    </ejs-grid>`,
  providers: [GroupService, FilterService, PageService]
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('Grid')
  public grid?: GridComponent;
  public message: string = '';
```

```

public persistedGridSettings?: object;
ngOnInit(): void {
    this.data = data;
}
save() {
    this.persistedGridSettings = JSON.parse(((this.grid as
GridComponent)).getPersistData());
    var gridColumns = Object.assign([], ((this.grid as
GridComponent)).getColumns());
    (this.persistedGridSettings as
any).columns.forEach((persistedColumn: Column) => {
        const column = gridColumns.find((col: Column) => col.field ===
persistedColumn.field);
        if (column) {
            persistedColumn.headerText = 'Text Changed';
            persistedColumn.template = (column as Column).template;
            persistedColumn.headerTemplate = (column as
Column).headerTemplate;
        }
    });
    window.localStorage.setItem('gridOrders1',
JSON.stringify(this.persistedGridSettings));
    this.grid?.setProperties(this.persistedGridSettings as object);
    this.message = 'Saved the headerText, template column, and
headerTemplate properties in the persisted settings';
}
restore() {
    const savedSettings = window.localStorage.getItem("gridOrders1");
    if (savedSettings) {
        this.grid?.setProperties(JSON.parse(savedSettings));
        this.message = 'Restored the headerText, template column, and
headerTemplate';
    } else {
        this.message = 'No saved settings found.';
    }
}
}

```

## MAIN.TS

```

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

```

## Toolbar in Angular Grid component

The toolbar in the Syncfusion Angular Grid component offers several general use cases to enhance data manipulation and overall experience. Actions such as adding, editing, and deleting records within the grid can be performed, providing efficient data manipulation capabilities. The toolbar also facilitates data export and import functionality, allowing you to generate downloadable files in formats like Excel, CSV, or PDF.

To enable the toolbar functionality, you need to inject the **ToolbarService** in the provider section of your **AppModule**. This service provides the necessary methods to interact with the toolbar items. The toolbar

can be customized with built-in toolbar items or custom toolbar items using the [toolbar](#) property. The `toolbar` property accepts an array of strings representing the built-in toolbar items or an array of [ItemModel](#) objects for custom toolbar items.

The following example demonstrates how to enable toolbar items in the grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='270px'
[toolbar]='toolbar'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = data;
    this.toolbar = ['Print', 'Search'];
  }
}
```

#### MAIN.TS

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

### Enable or disable toolbar items

Enabling or disabling toolbar items dynamically in Syncfusion Angular Grid is to provide control over the availability of specific functionality based on application logic. This feature allows you to customize the toolbar based on various conditions or individuals interactions.

You can enable or disable toolbar items dynamically by using the [enableToolbarItems](#) method. This method allows you to control the availability of specific toolbar items based on your application logic.

In the following example, the [EJ2 Toggle Switch Button component](#) is added to enable and disable the toolbar items using `enableToolbarItems` method. When the switch is toggled, the [change](#) event is triggered and the toolbar items are updated accordingly.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, ToolbarService, PageService } from
 '@syncfusion/ej2-angular-grids'
import {
  ButtonModule,
  CheckBoxModule,
  RadioButtonModule,
  SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
import { data } from './datasource';
import { GridComponent, GroupSettingsModel } from '@syncfusion/ej2-angular-
grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    GridModule,
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
  ],
  providers: [ToolbarService, PageService, GroupService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div>
      <label style="padding: 10px 10px">
        Enable or disable toolbar items
      </label>
      <ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-switch>
    </div>
    <ejs-grid id="Grid" #grid [dataSource]='data' height='200px'
    [allowGrouping]='true' [groupSettings]='groupOptions' [toolbar]='toolbar'
    (toolbarClick)="clickHandler($event)">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=100></e-
column>
        <e-column field='ShipName' headerText='Ship Name' width=120></e-
column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbar?: string[];
        public groupOptions?: GroupSettingsModel;
        public toolbarObj?: object[];
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbar = ['Expand', 'Collapse'];
            this.groupOptions = { columns: ['CustomerID'] };
        }
        clickHandler(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_Collapse') { // Grid_Collapse is control
id + '_' + toolbar value.
                (this.grid as GridComponent).groupModule.collapseAll();
            }
            if (args.item.id === 'Grid_Expand') {
                (this.grid as GridComponent).groupModule.expandAll();
            }
        }
        onSwitchChange(args: ChangeEventArgs) {
            if (args.checked) {
                (this.grid as
GridComponent).toolbarModule.enableItems(['Grid_Collapse', 'Grid_Expand'],
false); // Disable toolbar items.
            } else {
                (this.grid as
GridComponent).toolbarModule.enableItems(['Grid_Collapse', 'Grid_Expand'],
true); // Enable toolbar items.
            }
        }
    }
}

```

### MAIN.TS

```

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

```

### Add toolbar at the bottom of grid

By adding the toolbar at the bottom of the Syncfusion Angular Grid, important actions and functionality remain consistently visible and easily accessible, providing easy access to actions and operations without the need for scrolling.

To add the toolbar at the bottom of the Grid, you can utilize the [created](#) event. By handling this event, you can dynamically insert the toolbar items at the desired position in the grid layout.

The following example shows how to add the toolbar items at the bottom using `created` event of the grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PageService } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule,
    ButtonModule
  ],
  providers: [ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid id='Grid' #grid [dataSource]='data'
[toolbar]='toolbar' (created)="created()" height='200px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' type='number' isPrimaryKey='true' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
type='string' width=100></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' type='number' format='C2' width=80></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' type='date' format='yMd' width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbar?: ToolbarItems[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbar = ['Print', 'Search'];
  }
  created() {
    let toolbar = ((this.grid as GridComponent).element as
HTMLElement).querySelector('.e-toolbar');
    (this.grid as GridComponent).element.appendChild(toolbar as
HTMLElement);
  }
}
```



**MAIN.TS**

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

**Customize toolbar buttons using CSS**

Customizing toolbar buttons in Syncfusion Angular Grid using CSS involves modifying the appearance of built-in toolbar buttons by applying CSS styles. This provides a flexible and customizable way to enhance the visual presentation of the toolbar and create a cohesive interface.

The appearance of the built-in toolbar buttons can be modified by applying the following CSS styles.

```
`css
.e-grid .e-toolbar .e-tbar-btn .e-icons,
.e-grid .e-toolbar .e-toolbar-items .e-toolbar-item .e-tbar-btn {
background: #add8e6;
}
`
```

The following example demonstrates how to change the background color of the **Add**, **Edit**, **Delete**, **Update** and **Cancel** toolbar buttons by applying CSS styles

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService } from '@syncfusion/ej2-angular-grids'
import { ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [ToolbarService, EditService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='270px'
[toolbar]='toolbar' [editSettings]="editSettings">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
```

```

        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbar?: ToolbarItems[];
        public editSettings?: EditSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.editSettings = {
                allowEditing: true,
                allowAdding: true,
                allowDeleting: true,
            };
            this.toolbar = [
                'Add',
                'Edit',
                'Delete',
                'Update',
                'Cancel',
            ];
        }
    }
}

```

### 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](#)
- [How to add a router link in the toolbar in Angular Grid](#)
- [How to show or hide the delete button in the toolbar in Angular Grid](#)
- [How to display column as radio button in dialog editing in Angular Grid](#)

### Pdf export in Angular Grid component

The PDF export feature in the Syncfusion Angular Grid allows you to export grid data to a PDF document, providing the ability to generate printable reports or share data in a standardized format.

To enable PDF export in the grid, you need to set the [allowPdfExport](#) property to **true** and use the [pdfExport](#) method for exporting.

To use PDF export, inject the **PdfExportService** in the provider section of the **AppModule**.

The following example demonstrates how to perform a PDF export action in the grid.

### APP.COMPONENT.TS

```

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

```

```

import { GridModule, ToolbarService, PdfExportService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule

  ],
  providers: [PdfExportService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions'
      height='272px' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>`,
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['PdfExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
component id + _ + toolbar item name
      (this.grid as GridComponent).pdfExport();
    }
  }
}

```

## MAIN.TS

```

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

```

### Show spinner while exporting

Showing a spinner while exporting in the Syncfusion Angular Grid enhances the experience by displaying a spinner during the export process. This feature provides a visual indication of the export progress, improving the understanding of the exporting process.

To show or hide a spinner while exporting the grid, you can utilize the [showSpinner](#) and [hideSpinner](#) methods provided by the Grid within the [toolbarClick](#) event.

The `toolbarClick` event is triggered when a toolbar item in the Grid is clicked. Within the event handler, the code checks if the clicked **item** is related with PDF export, specifically the **Grid\_pdfexport** item. If a match is found, the `showSpinner` method is used on the Grid instance to display the spinner.

To hide the spinner after the exporting is completed, bind the [pdfExportComplete](#) event and use the `hideSpinner` method on the Grid instance to hide the spinner.

The following example demonstrates how to show and hide the spinner during PDF export in a grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [PdfExportService, ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[allowPaging]=true [toolbar]='toolbarOptions' height='272px'
[allowPdfExport]='true' (pdfExportComplete)='pdfExportComplete()'
(toolbarClick)='toolbarClick($event)'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ProductName' headerText='Product Name'
width=110></e-column>
      <e-column field='Quantity' headerText='Quantity' width=100></e-
column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
}
```

```

        this.toolbarOptions = ['PdfExport'];
    }
    toolbarClick(args: ClickEventArgs): void {
        if (args.item.id === 'Grid_pdfexport') {
            (this.grid as GridComponent).showSpinner();
            (this.grid as GridComponent).pdfExport();
        }
    }
    pdfExportComplete(): void {
        (this.grid as GridComponent).hideSpinner();
    }
}

```

### MAIN.TS

```

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

```

### Binding custom data source while exporting

The Syncfusion Angular Grid component provides a convenient way to export data to a PDF format. With the PDF export feature, you can define a custom data source while exporting. This allows you to export data that is not necessarily bind to the grid, which can be generated or retrieved based on your application logic.

To export data, you need to define the [dataSource](#) property within the [pdfExportProperties](#) object. This property represents the data source that will be used for the PDF export.

The following example demonstrates how to render custom data source during PDF export. By utilizing the [pdfExport](#) method and passing the `pdfExportProperties` object through the grid instance, the grid data will be exported to a PDF using the dynamically defined data source.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, ToolbarService, PdfExportService, PageService } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, changedData } from './datasource';
import { GridComponent, ToolbarItems, PdfExportProperties } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [PdfExportService, ToolbarService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data' [allowPaging]='true' [toolbar]='toolbarOptions'

```

```

        height='220px' [allowPaging]='true' [allowPdfExport]='true'
        (toolbarClick)='toolbarClick($event)'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=110></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
            </e-columns>
        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        @ViewChild('grid') public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['PdfExport'];
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_pdfexport') { // 'Grid_pdfexport' -> Grid
component id + _ + toolbar item name
                const pdfExportProperties: PdfExportProperties = {
                    dataSource: changedData,
                };
                (this.grid as GridComponent).pdfExport(pdfExportProperties);
            }
        }
    }
}

```

## MAIN.TS

```

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

```

Make sure to utilize the [field](#) property that you have declared in the grid columns when modifying the data source for exporting.

### Exporting with custom aggregate

Custom aggregates in the Syncfusion Angular Grid involves exporting grid data that includes additional calculated values based on specific requirements. This feature enables you to show the comprehensive view of the data in the exported file by incorporating the specific aggregated information you need for analysis or reporting purposes.

In order to utilize custom aggregation, you need to specify the [type](#) property as **Custom** and provide the custom aggregate function in the [customAggregate](#) property.

Within the **customAggregateFn** function, it takes an input data that contains a result property. The function calculates the count of objects in this data where the **ShipCountry** field value is equal to **Brazil** and returns the count with a descriptive label.

The following example shows how to export the grid with a custom aggregate that shows the calculation of the **Brazil** count of the **ShipCountry** column.

#### APP.COMPONENT.TS

```
{% raw %}
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems, ReturnType } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions' height='272px' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=100></e-column>
<e-column field='ShipCity' headerText='Ship City' width=100></e-column>
<e-column field='ShipCountry' headerText='ShipCountry' width=100></e-column>
</e-columns>
<e-aggregates>
<e-aggregate>
<e-columns>
<e-column
columnName="ShipCountry"
type="Custom"
[customAggregate]="customAggregateFn"
>
<ng-template #footerTemplate let-data> {{ data.Custom }}
</ng-template>
</e-column>
</e-columns>
</e-aggregate>
</e-aggregates>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['PdfExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_pdfexport') {
      (this.grid as GridComponent).pdfExport();
    }
  }
}
```

```

public customAggregateFn = (customData: object[] | { result?: object[] }) =>
{
  let data: object[] = [];
  if ('result' in customData && Array.isArray(customData.result)) {
    data = customData.result;
  } else if (Array.isArray(customData)) {
    data = customData;
  }
  const brazilCount = data.filter((item: object) => (item as
itemType)['ShipCountry'] === 'Brazil').length;
  return `Brazil count: ${brazilCount}`;
};
}
interface itemType {
  OrderID: number;
  CustomerID: string;
  EmployeeID: number;
  OrderDate: Date;
  ShipName: string;
  ShipCountry: string;
}
{% enddraw %}

```

## MAIN.TS

```

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

```

## Exporting with cell and row spanning

Exporting data from the Syncfusion Angular Grid with cell and row spanning enables you to maintain cell and row layout in the exported data. This feature is useful when you have merged cells or rows in the Grid and you want to maintain the same structure in the exported file.

To achieve this, you can utilize the [rowSpan](#) and [colSpan](#) properties in the [queryCellInfo](#) event of the Grid. This event allows you to define the span values for specific cells. Additionally, you can customize the appearance of the grid cells during the export using the [pdfQueryCellInfo](#) event of the Grid.

The following example demonstrates how to perform export with cell and row spanning using [queryCellInfo](#) and [pdfQueryCellInfo](#) events of the Grid.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService, AggregateService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems, PdfQueryCellInfoEventArgs,
QueryCellInfoEventArgs, Column } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

```



```

        GridModule
    ],
    providers: [PdfExportService, ToolbarService, AggregateService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #grid id='Grid' gridLines="Both"
[dataSource]='data' [toolbar]='toolbarOptions' height='272px'
[allowPdfExport]='true' (toolbarClick)='toolbarClick($event)'
(pdfQueryCellInfo)="pdfQueryCellInfo($event)"
(queryCellInfo)="queryCellInfoEvent($event)">
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
            <e-column field='Freight' headerText='Freight'
width=80></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            <e-column field='ShipCountry' headerText='Ship
Country' width=100></e-column>
        </e-columns>
    </ejs-grid>`
))
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    @ViewChild('grid') public grid?: GridComponent;
    ngOnInit(): void {
        this.data = data;
        this.toolbarOptions = ['PdfExport'];
    }
    toolbarClick(args: ClickEventArgs): void {
        if (args.item.id === 'Grid_pdfexport') {
            (this.grid as any).pdfExport();
        }
    }
    queryCellInfoEvent = function (args: QueryCellInfoEventArgs) {
        switch ((args.data as ColumnDataTypes).OrderID) {
            case 10248:
                if ((args.column as Column).field === 'CustomerID') {
                    args.rowSpan = 2;
                }
                break;
            case 10250:
                if ((args.column as Column).field === 'CustomerID') {
                    args.colSpan = 2;
                }
                break;
            case 10252:
                if ((args.column as Column).field === 'OrderID') {
                    args.rowSpan = 3;
                }
                break;
            case 10256:
                if ((args.column as Column).field === 'CustomerID') {

```

```

        args.colSpan = 3;
    }
    break;
case 10261:
    if ((args.column as Column).field === 'Freight') {
        args.colSpan = 2;
    }
    break;
}
}
pdfQueryCellInfo = function (args: PdfQueryCellInfoEventArgs) {
    switch ((args.data as columnDataType).OrderID) {
        case 10248:
            if ((args.column as Column).field === 'CustomerID') {
                (args.cell as PdfCell).rowSpan = 2;
            }
            break;
        case 10250:
            if ((args.column as Column).field === 'CustomerID') {
                args.colSpan = 2;
            }
            break;
        case 10252:
            if ((args.column as Column).field === 'OrderID') {
                (args.cell as PdfCell).rowSpan = 3;
            }
            break;
        case 10256:
            if ((args.column as Column).field === 'CustomerID') {
                args.colSpan = 3;
            }
            break;
        case 10261:
            if ((args.column as Column).field === 'Freight') {
                args.colSpan = 2;
            }
            break;
    }
};
}
interface columnDataType{
    field: number;
    OrderID:number,
    Freight:number,
    CustomerID:string,
    ShipCity:string,
    ShipName:string,
    ShipCountry:string,
    ShipPostalCode:number
}
interface PdfCell {
    rowSpan?: number;
}

```

**MAIN.TS**

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

\* The [updateCell](#) method does not support row and column spanning.

### Exporting with custom date format

The exporting functionality in the Syncfusion Angular Grid allows you to export grid data, including custom date format. This feature is useful when you need to export grid data with customized date values.

To apply a custom date format to grid columns during the export, you can utilize the [columns.format](#) property. This property allows you to define a custom format using format options.

The following example demonstrates how to export the grid data with custom date format. In this example, the formatOptions object is used as the `columns.format` property for the **OrderDate** column. This custom date format displays the date in the format of day-of-the-week, month abbreviation, day, and 2-digit year (e.g., Thu, Jul 4, '96).

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService, AggregateService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [PdfExportService, ToolbarService, AggregateService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions' height='272px' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='OrderDate' headerText='Order Date'
[format]='formatOptions' width=100></e-column>
      <e-column field='Freight' headerText='Freight'
width=80></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
```

```

public formatOptions?: object;
@ViewChild('grid') public grid?: GridComponent;
ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['PdfExport'];
    this.formatOptions = { type: 'date', format: "EEE, MMM d, 'yy" };
}
toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_pdfexport') {
        (this.grid as GridComponent).pdfExport();
    }
}
}

```

## MAIN.TS

```

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

```

## Exporting multiple grids

Exporting multiple grids in the Syncfusion Angular Grid component allows you to export different grids to compare them side by side in external applications on the same or different pages of a PDF file. Each grid is identified by its unique ID. You can specify which grid to export by listing their IDs in the [exportGrids](#) property.

### Same page

PDF exporting provides support for exporting multiple grids on the same page. To export the grids on the same page, define [multipleExport.type](#) as **AppendToPage** in [pdfExportProperties](#). It also has an option to provide blank space between the grids. This blank space can be defined by using [multipleExport.blankSpace](#) property.

The following example demonstrates how to export multiple grids to the same page in a PDF file when a toolbar item is clicked.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, ToolbarService, PdfExportService } from '@syncfusion/ej2-angular-grids';
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridComponent, ToolbarItems, PdfExportProperties } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
@Component({
    imports: [
        GridModule
    ],
    providers: [PdfExportService, ToolbarService],
    standalone: true,
    selector: 'app-root',

```

```

    template: `<p><b>First Grid:</b></p>
    <ejs-grid #firstGrid id='FirstGrid' [dataSource]='firstGridData'
    [toolbar]='toolbarOptions' [allowPdfExport]='true'
    (toolbarClick)='toolbarClick($event)' [exportGrids]='exportGrids'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=110></e-column>
        </e-columns>
    </ejs-grid>
    <p><b>Second Grid:</b></p>
    <ejs-grid #secondGrid id='SecondGrid'
[dataSource]='secondGridData' [allowPdfExport]='true'>
        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=90></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
            <e-column field='LastName' headerText='Last Name'
width=100></e-column>
            <e-column field='City' headerText='City' width=100></e-
column>
        </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public firstGridData?: object[];
    public secondGridData?: object[];
    public toolbarOptions?: ToolbarItems[];
    public exportGrids?: string[];
    @ViewChild('firstGrid') public firstGrid?: GridComponent;
    @ViewChild('secondGrid') public secondGrid?: GridComponent;
    ngOnInit(): void {
      this.firstGridData = data;
      this.secondGridData = employeeData;
      this.toolbarOptions = ['PdfExport'];
      this.exportGrids = ['FirstGrid', 'SecondGrid']
    }
    toolbarClick = (args: ClickEventArgs) => {
      if (args.item.id === 'FirstGrid_pdfexport') { // 'Grid_pdfexport' ->
Grid component id + _ + toolbar item name
        const appendPdfExportProperties: PdfExportProperties = {
          multipleExport: { type: "AppendToPage", blankSpace: 10 }
        };
        (this.firstGrid as
GridComponent).pdfExport(appendPdfExportProperties, true);
      }
    }
  }
}

```

**MAIN.TS**

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

### [New page](#)

PDF export functionality enables the exporting of multiple grids into separate pages (each grid on a new page) within the PDF file.

To achieve this, you can follow these steps:

1. Access the [pdfExportProperties](#) of the Grid component.
2. Set the [multipleExport.type](#) property to **NewPage**.
3. Trigger the PDF export operation.

The following example demonstrates how to export multiple grids to a PDF file when a toolbar item is clicked.

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridComponent, ToolbarItems, PdfExportProperties } from
 '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [PdfExportService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<p><b>First Grid:</b></p>
<ejs-grid #firstGrid id='FirstGrid' [dataSource]='firstGridData'
[toolbar]='toolbarOptions' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)' [exportGrids]='exportGrids'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
    <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
    <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
    <e-column field='ShipName' headerText='Ship Name'
width=110></e-column>
  </e-columns>
</ejs-grid>
<p><b>Second Grid:</b></p>
<ejs-grid #secondGrid id='SecondGrid'
[dataSource]='secondGridData' [allowPdfExport]='true'>
```

```

        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=90></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
            <e-column field='LastName' headerText='Last Name'
width=100></e-column>
            <e-column field='City' headerText='City' width=100></e-
column>
        </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
    public firstGridData?: object[];
    public secondGridData?: object[];
    public toolbarOptions?: ToolbarItems[];
    public exportGrids?: string[];
    @ViewChild('firstGrid') public firstGrid?: GridComponent;
    @ViewChild('secondGrid') public secondGrid?: GridComponent;
    ngOnInit(): void {
        this.firstGridData = data;
        this.secondGridData = employeeData;
        this.toolbarOptions = ['PdfExport'];
        this.exportGrids = ['FirstGrid', 'SecondGrid'];
    }
    toolbarClick = (args: ClickEventArgs) => {
        const appendPdfExportProperties: PdfExportProperties = {
            multipleExport: { type: 'NewPage' }
        };
        if (args.item.id === 'FirstGrid_pdfexport') { // 'Grid_pdfexport' ->
Grid component id + _ + toolbar item name
            (this.firstGrid as
GridComponent).pdfExport(appendPdfExportProperties, true);
        }
    }
}
}

```

## MAIN.TS

```

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

```

## Limitations

- Multiple grids exporting feature is not supported with server side exporting.

## Exporting hierarchy grid

Exporting a hierarchy grid in the Syncfusion Angular Grid component allows you to generate a PDF document that includes the master grid along with its child grids. This feature is useful when you need to export hierarchical data with its related details.

To achieve this, you can customize the exporting behavior by using the `pdfExportProperties.hierarchyExportMode` property of the Grid.

The `hierarchyExportMode` property allows you to specify the exporting behavior for the hierarchy grid. The following options are available:

Mode	Behavior
Expanded	Exports the master grid with expanded child grids.
All	Exports the master grid with all child grids, expanded or not.
None	Exports only the master grid without any child grids.

The following example demonstrates how to export hierarchical grid to PDF document. Also change the `pdfExportProperties.hierarchyExportMode` property by using `value` property of the `DropDownList` component.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService, DetailRowService }
  from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import {
  GridComponent,
  ToolbarItems,
  PdfExportProperties,
  GridModel,
} from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
  providers: [PdfExportService, ToolbarService, DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0"> Change the hierarchy export
mode: </label>
      <ejs-dropdownlist
style="margin-top: 5px"
#ddlData
index="0"
width="150"
[dataSource]="ddlData"></ejs-dropdownlist>
    </div>
```



```

<ejs-grid #grid id='Grid' [dataSource]='data' [toolbar]='toolbarOptions'
[childGrid]='childGrid'
  height='220px' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)'>
  <e-columns>
    <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right' width=90></e-column>
    <e-column field='FirstName' headerText='FirstName'
width=100></e-column>
    <e-column field='LastName' headerText='Last Name' width=100></e-
column>
    <e-column field='City' headerText='City' width=100></e-column>
  </e-columns>
</ejs-grid>`
))
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  @ViewChild('dropDownList')
  public dropDownList?: DropDownListComponent;
  public ddlData: object[] = [
    { text: 'Expanded', value: 'Expanded' },
    { text: 'All', value: 'All' },
    { text: 'None', value: 'None' },
  ];
  public childGrid: GridModel = {
    dataSource: data,
    queryString: 'EmployeeID',
    columns: [
      {
        field: 'OrderID',
        headerText: 'Order ID',
        textAlign: 'Right',
        width: 90,
      },
      { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
      { field: 'ShipCity', headerText: 'Ship City', width: 100 },
      { field: 'ShipName', headerText: 'Ship Name', width: 110 },
    ],
  };
  ngOnInit(): void {
    this.data = employeeData;
    this.toolbarOptions = ['PdfExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_pdfexport') {
      // 'Grid_pdfexport' -> Grid component id + _ + toolbar item name
      const exportProperties: PdfExportProperties = {
        hierarchyExportMode: (this.dropDownList as
DropDownListComponent).value as PdfExportProperties["hierarchyExportMode"],
      };
      (this.grid as GridComponent).pdfExport(exportProperties);
    }
  }
}

```

**MAIN.TS**

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

**Remove header row while exporting**

When exporting data from the Syncfusion Angular Grid, you have an option to remove the header row from the exported file. This can be useful when you want to export grid data without including the header values in the exported document.

To achieve this, you can utilize the [pdfHeaderQueryCellInfo](#) event of the Grid. This event allows you to customize the header cells during the PDF export process. By handling this event, you can remove the header row from the exported file by not providing any content and height for the header cells. This ensures that the exported file contains only the data rows without including the header information.

The following example demonstrates how to perform export without header using [pdfHeaderQueryCellInfo](#) event of the Grid.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService, AggregateService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, Cell } from './datasource';
import { GridComponent, ToolbarItems, PdfHeaderQueryCellInfoEventArgs } from
 '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [PdfExportService, ToolbarService, AggregateService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions' height='272px' [allowPdfExport]='true'
(toolbarClick)='toolbarClick($event)'
(pdfHeaderQueryCellInfo)="pdfHeaderQueryCellInfo($event)">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipCountry' headerText='Ship
Country' width=100></e-column>
    </e-columns>
  </ejs-grid>`
```

```

    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        @ViewChild('grid') public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['PdfExport'];
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_pdfexport') {
                (this.grid as GridComponent).pdfExport();
            }
        }
        pdfHeaderQueryCellInfo({cell}: PdfHeaderQueryCellInfoEventArgs) {
            const typedCell = cell as Cell;
            typedCell.value = '';
            if (typedCell.value === '') {
                typedCell.height = '';
            }
        }
    }
}

```

### 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

- [Exporting Grid in Cordova application](#)

### Excel exporting in Angular Grid component

The Excel or CSV exporting feature in the Angular Grid component allows you to export the Grid data to an Excel or CSV document. This can be useful when you need to share or analyze the data in a spreadsheet format.

To enable Excel export in the Grid component, you need to set the [allowExcelExport](#) property to **true**. This property is responsible for enabling the Excel or CSV export option in the Grid.

To initiate the excel export process, you need to use the [excelExport](#) method provided by the Grid component. This method is responsible for exporting the Grid data to an Excel document.

To use the Excel or CSV export feature, you need to inject the **ExcelExportService** in the provider section of your **AppModule**. This allows the Grid component to access the necessary services for exporting data to Excel or CSV.

To initiate the CSV export process, you need to use the [csvExport](#) method provided by the Grid component. This method is responsible for exporting the Grid data to an CSV document.

The following example demonstrates how to perform a Excel or CSV export action in the grid:

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions'
height='272px' [allowExcelExport]='true'
(toolbarClick)='toolbarClick($event)'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150>
      </e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['ExcelExport', 'CsvExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_excelexport') {
      // 'Grid_excelexport' -> Grid component id + _ + toolbar item
      name
      (this.grid as GridComponent).excelExport();
    }
    else if (args.item.id === 'Grid_csvexport') {
      // 'Grid_csvexport' -> Grid component id + _ + toolbar item name
      (this.grid as GridComponent).csvExport();
    }
  }
}

```

**MAIN.TS**

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

**Show spinner while exporting**

Showing a spinner while exporting in the Grid enhances the experience by displaying a spinner during the export process. This feature provides a visual indication of the export progress, improving the understanding of the exporting process.

To show or hide a spinner while exporting the grid, you can utilize the [showSpinner](#) and [hideSpinner](#) methods provided by the Grid within the `toolbarClick` event.

The `toolbarClick` event is triggered when a toolbar item in the Grid is clicked. Within the event handler, the code checks if the clicked **item** is related with Excel or CSV export, specifically the **Gridexcelexport** or **Gridcsvexport** item. If a match is found, the `showSpinner` method is used on the Grid instance to display the spinner.

To hide the spinner after the exporting is completed, bind the [excelExportComplete](#) event and use the `hideSpinner` method on the Grid instance to hide the spinner.

The following example demonstrates how to show and hide the spinner during Excel export in a grid.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, ToolbarItems } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [ExcelExportService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
    [toolbar]='toolbarOptions' height='272px'
    [allowExcelExport]='true'
    (excelExportComplete)='excelExportComplete()'
    (toolbarClick)='toolbarClick($event)'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
        textAlign='Right'
        width='90'></e-column>
      <e-column field='ProductName' headerText='Product
        Name' width='100'>
    </e-column>
```

```

        <e-column field='ProductID' headerText='Product ID'
textAlign='Right'
        width='80'></e-column>
        <e-column field='CustomerName' headerText='Customer
Name' width='120'>
        </e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        @ViewChild('grid') public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['ExcelExport'];
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_excelexport') {
                (this.grid as GridComponent).showSpinner();
                (this.grid as GridComponent).excelExport();
            }
        }
        excelExportComplete(): void {
            (this.grid as GridComponent).hideSpinner();
        }
    }
}

```

## MAIN.TS

```

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

```

### Binding custom data source while exporting

The Grid component provides a convenient way to export data to a Excel or CSV format. With the Excel or CSV export feature, you can define a custom data source while exporting. This allows you to export data that is not necessarily bind to the grid, which can be generated or retrieved based on your application logic.

To export data, you need to define the [dataSource](#) property within the [excelExportProperties](#) object. This property represents the data source that will be used for the Excel or CSV export.

The following example demonstrates how to render custom dataSource during Excel export. By calling the [excelExport](#) method and passing the [excelExportProperties](#) object through the grid instance, the grid data will be exported to a Excel using the dynamically defined data source.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';

```

```

import { data } from './datasource';
import { GridComponent, ToolbarItems, ExcelExportProperties } from
 '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions'
      height='272px' [allowExcelExport]='true'
      (toolbarClick)='toolbarClick($event)'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
      width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150>
      </e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['ExcelExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
    if (args.item.id === 'Grid_excelexport') {
      // 'Grid_excelexport' -> Grid component id + _ + toolbar item
name
      const excelExportProperties: ExcelExportProperties = {
        dataSource: data
      };
      (this.grid as GridComponent).excelExport(excelExportProperties);
    }
  }
}

```

## MAIN.TS

```

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

```

### Exporting with custom aggregate

Exporting grid data with custom aggregates allows you to include additional calculated values in the exported file based on specific requirements. This feature is highly valuable for providing a comprehensive view of the data in the exported file, incorporating specific aggregated information for analysis or reporting purposes.

In order to utilize custom aggregation, you need to specify the [type](#) property as **Custom** and provide the custom aggregate function in the [customAggregate](#) property.

Within the **customAggregateFn** function, it takes an input data that contains a result property. The function calculates the count of objects in this data where the **ShipCountry** field value is equal to **Brazil** and returns the count with a descriptive label.

The following example shows how to export the grid with a custom aggregate that shows the calculation of the **Brazil** count of the **ShipCountry** column.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, AggregateService }
from '@syncfusion/ej2-angular-grids'
import { Component, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, AggregateService],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <ejs-grid #grid id="DefaultExport" [dataSource]="data"
[toolbar]="toolbar"
      (toolbarClick)="toolbarClick($event)"
[allowExcelExport]="true">
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='right' width=120>
        </e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field="Freight" headerText="Freight" width="150"
format="C2"
          textAlign="Right"></e-column>
        <e-column field="OrderDate" headerText="Order Date"
width="150"
          format="yMd" textAlign="Right"></e-column>
        <e-column field="ShipCountry" headerText="Ship Country"
width="150">
          </e-column>
        </e-columns>
      </e-column>
    </e-columns>
  </div>`
})
export class AppComponent {
  toolbar: string[] = [
    { text: 'Export', click: () => { } }
  ];
  toolbarClick(event: ClickEventArgs): void {
    console.log('Toolbar Clicked');
  }
}
```



```

        <e-aggregates>
            <e-aggregate>
                <e-columns>
                    <e-column columnName="ShipCountry" type="Custom"
                        [customAggregate]="customAggregateFn">
                        <ng-template #footerTemplate let-data> {{
data.Custom }}</ng-template>
                    </e-column>
                </e-columns>
            </e-aggregate>
        </e-aggregates>
    </ejs-grid>
</div>`
    })
    export class AppComponent {

        public data?: Object[];
        public toolbar?: string[];
        @ViewChild('grid')
        public grid?: GridComponent;
        public ngOnInit(): void {
            this.data = data.slice(0, 20);
            this.toolbar = ['ExcelExport'];
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id=='DefaultExport_exceleexport') {
                (this.grid as GridComponent).excelExport();
            }
        }
        public customAggregateFn = (customData: any) => {
            const brazilCount=customData.result ? customData.result.filter(
                (item: object) => (item as any)['ShipCountry'] === 'Brazil'
            ).length:
            customData.filter(
                (item: object) => (item as any)['ShipCountry'] === 'Brazil'
            ).length;
            return `Brazil Count::${brazilCount}`;
        };
    }
}

```

## MAIN.TS

```

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

```

## Exporting with cell and row spanning

Exporting data from the Grid with cell and row spanning enables you to maintain cell and row layout in the exported data. This feature is useful when you have merged cells or rows in the Grid and you want to maintain the same structure in the exported file.

To achieve this, you can utilize the [rowSpan](#) and [colSpan](#) properties in the `queryCellInfo` event of the Grid. This event allows you to define the span values for specific cells. Additionally, you can customize the appearance of the grid cells during the export using the [excelQueryCellInfo](#) event of the Grid.

The following example demonstrates how to perform export with cell and row spanning using `queryCellInfo` and `excelQueryCellInfo` events of the Grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
  from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, columnDataType } from './datasource';
import { GridComponent, ToolbarItems, ExcelQueryCellInfoEventArgs,
  QueryCellInfoEventArgs, Column, ExcelCell } from '@syncfusion/ej2-angular-
  grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';

@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions' height='272px'
      allowExcelExport='true' (toolbarClick)='toolbarClick($event)'
      (excelQueryCellInfo)="excelQueryCellInfo($event)"
      (queryCellInfo)="queryCellInfoEvent($event)">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field='Freight' headerText='Freight'
width=80></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipCountry' headerText='Ship
Country' width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbarOptions?: ToolbarItems[];
  public message?: string;
  @ViewChild('grid') public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.toolbarOptions = ['ExcelExport'];
  }
  toolbarClick(args: ClickEventArgs): void {
```

```

        if (args.item.id === 'Grid_excelexport') {
            (this.grid as GridComponent).excelExport();
        }
    }
    queryCellInfoEvent = function (args: QueryCellInfoEventArgs) {
        switch ((args.data as columnDataType).OrderID) {
            case 10248:
                if ((args.column as Column).field === 'CustomerID') {
                    args.rowSpan = 2;
                }
                break;
            case 10250:
                if ((args.column as Column).field === 'CustomerID') {
                    args.colSpan = 2;
                }
                break;
            case 10252:
                if ((args.column as Column).field === 'OrderID') {
                    args.rowSpan = 3;
                }
                break;
            case 10256:
                if ((args.column as Column).field === 'CustomerID') {
                    args.colSpan = 3;
                }
                break;
            case 10261:
                if ((args.column as Column).field === 'Freight') {
                    args.colSpan = 2;
                }
                break;
        }
    }
    excelQueryCellInfo = function
    ({data, cell, colSpan, column}: ExcelQueryCellInfoEventArgs) {
        switch ((data as columnDataType).OrderID) {
            case 10248:
                if ((column as Column).field === 'CustomerID') {
                    (cell as ExcelCell).rowSpan = 2;
                }
                break;
            case 10250:
                if ((column as Column).field === 'CustomerID') {
                    colSpan = 2;
                }
                break;
            case 10252:
                if ((column as Column).field === 'OrderID') {
                    (cell as ExcelCell).rowSpan = 3;
                }
                break;
            case 10256:
                if ((column as Column).field === 'CustomerID') {
                    (cell as ExcelCell).colSpan = 3;
                }
                break;
            case 10261:

```

```

        if ((column as Column).field === 'Freight') {
            (cell as ExcelCell).colSpan = 2;
        }
        break;
    }
};
}

```

### MAIN.TS

```

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

```

- The [updateCell](#) method does not support row and column spanning.

### Exporting with custom date format

The exporting functionality in the Syncfusion Angular Grid allows you to export grid data, including custom date format. This feature is useful when you need to export grid data with customized date values.

To apply a custom date format to grid columns during the export, you can utilize the [columns.format](#) property. This property allows you to define a custom format using format options.

The following example demonstrates how to export the grid with custom date format. In the example, the `formatOptions` object is used as the `format` property for the **OrderDate** column. This custom date format displays the date in the format of day-of-the-week, month abbreviation, day, and 2-digit year (e.g., Sun, May 8, '23).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-navigations'
@Component({
  imports: [
    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<div class="control-section">
    <ejs-grid #grid [dataSource]='data' allowPaging='true'
      [pageSettings]='pageSettings' [toolbar]='toolbar'
      (toolbarClick)='toolbarClick($event)'
      [allowExcelExport]='true'
      [allowPdfExport]='true'>

```

```

        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
width='120'
            textAlign='Right' isPrimaryKey='true'></e-
column>
            <e-column field='OrderDate' headerText='Order
Date' width='130'
            [format]="formatOption" textAlign='Right'></e-
column>
            <e-column field='CustomerID'
headerText='Customer ID' width='120'>
            </e-column>
            <e-column field='Freight' headerText='Freight'
[allowGrouping]="false"
            width='120' format='C2' textAlign='Right'></e-
column>
            <e-column field='ShipCountry' headerText='Ship
Country' width='150'>
            </e-column>
        </e-columns>
    </ejs-grid>
</div>`
    })
    export class AppComponent implements OnInit {
        public data?: Object[];
        public toolbar?: string[];
        public pageSettings?: Object;
        public refresh?: Boolean;
        public formatOption?: Object;
        @ViewChild('grid')
        public grid?: GridComponent;
        public ngOnInit(): void {
            this.data = data;
            this.formatOption = { type: 'date', format:"EEE, MMM d, 'yy"};
            this.toolbar = ['ExcelExport'];
            this.pageSettings = { pageCount: 5 };
        }
        toolbarClick(args: ClickEventArgs): void {
            if(args.item.text === 'Excel Export') {
                (this.grid as GridComponent).excelExport();
            }
        }
    }
}

```

## MAIN.TS

```

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

```

## Exporting multiple grids

Exporting multiple grids in the Syncfusion Angular Grid component allows you to export different grids to compare them side by side in external applications on the same or different pages of a Excel. Each

grid is identified by its unique ID. You can specify which grid to export by listing their IDs in the [exportGrids](#) property.

#### Same sheet

Excel exporting provides support for exporting multiple grids on the same page. This feature is particularly useful when you want to combine and organize data from different grids for a unified view in the exported Excel file.

To achieve this, you need to define the [multipleExport.type](#) property as **AppendToSheet** in the [excelExportProperties](#) object. This setting ensures that the data from each grid will be appended to the same Excel sheet.

Additionally, you have an option to include blank rows between the data of each grid to visually separate them in the exported Excel sheet. The number of blank rows to be inserted can be defined using the [multipleExport.blankRows](#) property.

The following example demonstrates how to export multiple grids to the same page in a Excel file when a toolbar item is clicked.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
  from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridComponent, ToolbarItems, ExcelExportProperties } from
  '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<p><b>First Grid:</b></p>
    <ejs-grid #grid1 id='FirstGrid' [dataSource]='firstData'
      [toolbar]='toolbarOptions' [allowExcelExport]='true'
      (toolbarClick)='toolbarClick($event)'
    [exportGrids]='exportGrids'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
          textAlign='Right'
          width=120></e-column>
        <e-column field='CustomerID' headerText='Customer
          ID' width=150>
        </e-column>
        <e-column field='ShipCity' headerText='Ship City'
          width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
          width=150></e-column>
      </e-columns>
    </ejs-grid>
    <p><b>Second Grid:</b></p>`
})
```

```

        <ejs-grid #grid2 id='SecondGrid'
[dataSource]='secondData'
        [allowExcelExport]='true'>
        <e-columns>
            <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right'
                width=120></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
            <e-column field='LastName' headerText='Last Name'
width=150></e-column>
            <e-column field='City' headerText='City'
width=150></e-column>
        </e-columns>
        </ejs-grid> `
    })
    export class AppComponent implements OnInit {
        public firstData?: object[];
        public secondData?: object[];
        public toolbarOptions?: ToolbarItems[];
        public exportGrids?: string[];
        @ViewChild('grid1') public firstGrid?: GridComponent;
        @ViewChild('grid2') public secondGrid?: GridComponent;
        ngOnInit(): void {
            this.firstData = data.slice(0, 5);
            this.secondData = employeeData.slice(0, 5);
            this.toolbarOptions = ['ExcelExport'];
            this.exportGrids = ['FirstGrid', 'SecondGrid']
        }
        toolbarClick = (args: ClickEventArgs) => {
            if (args.item.id === 'FirstGrid_excelexport') {
                // 'Grid_excelexport' -> Grid component id + _ + toolbar item
name
                const appendExcelExportProperties: ExcelExportProperties = {
                    multipleExport: { type: 'AppendToSheet', blankRows: 2 }
                };
                (this.firstGrid as
GridComponent).excelExport(appendExcelExportProperties, true);
            }
        }
    }
}

```

### 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, **multipleExport.blankRows** value is 5.

#### [New sheet](#)

Excel export functionality enables the exporting of multiple grids into separate pages (each grid on a new page) within the Excel file.

To achieve this, you can follow these steps:

1. Access the [excelExportProperties](#) of the Grid component.
2. Set the [multipleExport.type](#) to **NewPage**.
3. Trigger the Excel export operation.

The following example demonstrates how to export multiple grids to a Excel file when a toolbar item is clicked.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from './datasource';
import { GridComponent, ToolbarItems, ExcelExportProperties } from
 '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<p><b>First Grid:</b></p>
    <ejs-grid #grid1 id='FirstGrid' [dataSource]='firstData'
[toolbar]='toolbarOptions'
    [allowExcelExport]='true'
(toolbarClick)='toolbarClick($event)'
    [exportGrids]='exportGrids'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
        width=120></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=150>
        </e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
      </e-columns>
    </ejs-grid>
    <p><b>Second Grid:</b></p>
    <ejs-grid #grid2 id='SecondGrid'
[dataSource]='secondData'
    [allowExcelExport]='true'>
      <e-columns>
        <e-column field='EmployeeID' headerText='Employee
ID' textAlign='Right'
        width=120></e-column>
```



```

        <e-column field='FirstName' headerText='FirstName'
width=150>
        </e-column>
        <e-column field='LastName' headerText='Last Name'
width=150>
        </e-column>
        <e-column field='City' headerText='City'
width=150></e-column>
    </e-columns>
</ejs-grid>
})
export class AppComponent implements OnInit {
    public firstData?: object[];
    public secondData?: object[];
    public toolbarOptions?: ToolbarItems[];
    public exportGrids?: string[];
    @ViewChild('grid1') public firstGrid?: GridComponent;
    @ViewChild('grid2') public secondGrid?: GridComponent;
    ngOnInit(): void {
        this.firstData = data.slice(0, 5);
        this.secondData = employeeData.slice(0, 5);
        this.toolbarOptions = ['ExcelExport'];
        this.exportGrids = ['FirstGrid', 'SecondGrid']
    }
    toolbarClick = (args: ClickEventArgs) => {
        if (args.item.id === 'FirstGrid_excelexport') {
            // 'Grid_excelexport' -> Grid component id + _ + toolbar item
name
            const appendExcelExportProperties: ExcelExportProperties = {
                multipleExport: { type: 'NewSheet' }
            };
            (this.firstGrid as
GridComponent).excelExport(appendExcelExportProperties, true);
        }
    }
}

```

## MAIN.TS

```

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

```

## Limitations

- Multiple grids exporting feature is not supported with server side exporting.

## Exporting hierarchy grid

Exporting a hierarchy grid in the Syncfusion Angular Grid component allows you to generate a Excel or CSV document that includes the parent grid along with its child grids. This feature is useful when you need to export hierarchical data with its related details.

To achieve this, you can customize the exporting behavior by using the `ExcelExportProperties.hierarchyExportMode` property of the Grid. This property allows you to specify the exporting behavior for the hierarchy grid. The following options are available:

Mode	Behavior
----- -----	
Expanded	Exports the master grid with expanded child grids.
All	Exports the master grid with all child grids, expanded or not.
None	Exports only the master grid without any child grids.

The following example demonstrates how to export hierarchical grid to Excel document. Also change the `excelExportProperties.hierarchyExportMode` property by using `value` property of the DropDownList component:

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, DetailRowService }
from '@syncfusion/ej2-angular-grids'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, employeeData } from '../datasource';
import {
  GridComponent,
  ToolbarItems,
  ExcelExportProperties,
  GridModel,
} from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
import { DropDownListComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    GridModule,
    DropDownListAllModule
  ],
  providers: [ExcelExportService, ToolbarService, DetailRowService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <label style="padding: 10px 10px 26px 0">
        Change the hierarchy export mode: </label>
      <ejs-dropdownlist
        style="margin-top: 5px"
        #dropDownList
        index="0"
        width="150"
        [dataSource]="dropDownData"></ejs-dropdownlist>
    </div>
    <ejs-grid #grid id='Grid' [dataSource]='data'
    [toolbar]='toolbarOptions'
    [childGrid]='childGrid' height='220px' [allowExcelExport]='true'`
})
export class AppComponent implements OnInit {
  @ViewChild(GridComponent) grid: GridComponent;
  @ViewChild(DropDownListComponent) dropDownList: DropDownListComponent;

  ngOnInit(): void {
    this.grid.excelExportProperties.hierarchyExportMode = 'Expanded';
  }
}
```

```

        (toolbarClick)='toolbarClick($event) '>
        <e-columns>
            <e-column field='EmployeeID' headerText='Employee ID'
textAlign='Right'
            width=90></e-column>
            <e-column field='FirstName' headerText='FirstName'
width=100>
            </e-column>
            <e-column field='LastName' headerText='Last Name'
width=100>
            </e-column>
            <e-column field='City' headerText='City' width=100></e-
column>
        </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    @ViewChild('grid') public grid?: GridComponent;
    @ViewChild('dropDownList')
    public dropDownList?: DropDownListComponent;
    public dropDownData: object[] = [
        { text: 'None', value: 'None' },
        { text: 'Expanded', value: 'Expanded' },
        { text: 'All', value: 'All' },
    ];
    public childGrid: GridModel = {
        dataSource: data,
        queryString: 'EmployeeID',
        columns: [
            {
                field: 'OrderID',
                headerText: 'Order ID',
                textAlign: 'Right',
                width: 90,
            },
            { field: 'CustomerID', headerText: 'Customer ID', width: 100 },
            { field: 'ShipCity', headerText: 'Ship City', width: 100 },
            { field: 'ShipName', headerText: 'Ship Name', width: 110 },
        ],
    };
    ngOnInit(): void {
        this.data = employeeData;
        this.toolbarOptions = ['ExcelExport'];
    }
    toolbarClick(args: ClickEventArgs): void {
        if (args.item.id === 'Grid_excelexport') {
            // 'Grid_Excelexport' -> Grid component id + _ + toolbar item
name
            const exportProperties: ExcelExportProperties = {
                hierarchyExportMode: ((this.dropDownList as
DropDownListComponent).value as ExcelExportProperties["hierarchyExportMode"])
            );
            (this.grid as GridComponent).excelExport(exportProperties);
        }
    }
}

```

```
}
}
```

### MAIN.TS

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

### Limitations

- Microsoft Excel permits up to seven nested levels in outlines. So that in the grid we can able to provide only up to seven nested levels and if it exceeds more than seven levels then the document will be exported without outline option. Please refer the [Microsoft Limitation](#).

### Remove header row while exporting

When exporting data from the Syncfusion Angular Grid, you have an option to remove the header row from the exported file. This can be useful when you want to export grid data without including the header values in the exported document. To achieve this, you can utilize the [excelHeaderQueryCellInfo](#) and [created](#) event.

The following example demonstrates how to perform an export without the header by using the [excelHeaderQueryCellInfo](#) event to clear cell content in the header row and the [created](#) event to remove the header row from the Grid:

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { ExcelCell, ExcelHeaderQueryCellInfoEventArgs, GridComponent,
ToolbarItems, ExcelExport } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='Grid' [dataSource]='data'
[toolbar]='toolbarOptions'
height='272px' [allowExcelExport]='true'
(toolbarClick)='toolbarClick($event)'
(excelHeaderQueryCellInfo)="excelHeaderQueryCellInfo($event)"
(created)=created()>
<e-columns>
<e-column field='OrderID' headerText='Order ID'
textAlign='Right'`
```

```

        width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=100></e-column>
        <e-column field='OrderDate' headerText='Order Date'
        [format]='formatOptions' width=100></e-column>
        <e-column field='Freight' headerText='Freight'
width=80></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        public formatOptions?: object;
        @ViewChild('grid') public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.toolbarOptions = ['ExcelExport'];
            this.formatOptions = { type: 'date', format: "yMd" };
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_excelexport') {
                (this.grid as GridComponent).excelExport();
            }
        }
        excelHeaderQueryCellInfo(args: ExcelHeaderQueryCellInfoEventArgs) {
            (args.gridCell as ExcelCell).value = '';
        }
        created() {
            // var gridObj = (document.getElementById("Grid")).ej2_instances[0];
            var processGridExportObject =
                ((this.grid as GridComponent).excelExportModule as
any).__proto__.processGridExport;
            ((this.grid as GridComponent).excelExportModule as
any).__proto__.processGridExport = function (
                gobj:object,
                props:ExcelExport,
                r:boolean
            ) {
                var rows = processGridExportObject.call(this, gobj, props, r);
                rows.shift();
                rows.forEach((item:any, index:number) => {
                    item.index = index + 1;
                });
                return rows;
            };
        }
    }
}

```

## 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 formula for the cell while exporting

The Grid component provides a convenient way to add formulas to cells during the export process. This feature allows you to perform calculations and apply formulas to specific cells in the exported Excel document. This can be particularly useful when you need to include calculated values or perform complex calculations.

To add formulas to cells during the export process, you can utilize the [valueAccessor](#) method along with the [excelQueryCellInfo](#) event.

In the following example, the [toolbarClick](#) function handles a toolbar button click event. When the Excel Export button is clicked, it triggers the Excel export process. Inside this function, an [excelExportProperties](#) object is defined, specifying that hidden columns should be included in the export. Inside the [excelQueryCellInfo](#) event, the [valueAccessor](#) method generates formulas for the desired cells and assigns these formulas to the cell's formula property, ensuring that the calculated values are exported to the Excel document:

excelExportProperties

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, ExcelExportService, FilterService }
  from '@syncfusion/ej2-angular-grids'
import { Component, ViewChild } from '@angular/core';
import { inventoryData } from '../datasource';
import { GridComponent, ToolbarItems, ExcelExportProperties,
  ExcelQueryCellInfoEventArgs } from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    GridModule
  ],
  providers: [ExcelExportService, ToolbarService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `
    <ejs-grid #grid id='Grid' [dataSource]='data' [allowPaging]='true'
      [toolbar]='toolbarOptions' height='272px'
      [allowExcelExport]='true'
      (toolbarClick)='toolbarClick($event)'
      (excelQueryCellInfo)="excelQueryCellInfo($event)">
      <e-columns>
        <e-column field="Inventor" headerText="Inventor Name"
width="180" textAlign="Right">
        </e-column>
        <e-column field="NumberofPatentFamilies" headerText="Number of
Patent Families"
width="180" textAlign="Right"></e-column>
        <e-column field="Country" headerText="Country" width="140"
textAlign="Left">
        </e-column>
      </e-columns>
    </ejs-grid>
  `
})
```

```

        <e-column field="Mainfieldsofinvention" headerText="Main
fields of invention"
        width="200" textAlign="Left"></e-column>
        <e-column field="Number of INPADOC patents" headerText="Number
of INPADOC patents"
        width="180" textAlign="Right"></e-column>
        <e-column field="TotalPatents" headerText="Total Patents"
[visible]='false'
        [valueAccessor]="valueAccess" width="120"
textAlign="Right"></e-column>
        </e-columns>
    </ejs-grid>`
    })
    export class AppComponent {
        public data?: object[];
        public toolbarOptions?: ToolbarItems[];
        @ViewChild('grid') public grid?: GridComponent;
        public ngOnInit(): void {
            this.data = inventoryData;
            this.toolbarOptions = ['ExcelExport'];
        }
        toolbarClick(args: ClickEventArgs): void {
            if (args.item.id === 'Grid_excelexport') {
                // 'Grid_pdfexport' -> Grid component id + _ + toolbar item name
                const excelExportProperties: ExcelExportProperties = {
                    includeHiddenColumn: true,
                };
                (this.grid as GridComponent).excelExport(excelExportProperties);
            }
        }
        valueAccess = (field: string, data: Object[]) => {
            const cell = (this.data as Object[]).indexOf(data) + 2;
            return 'E' + cell + '+' + 'B' + cell;
        }
        excelQueryCellInfo(args: ExcelQueryCellInfoEventArgs): void {
            if (args.column.field === 'TotalPatents') {
                args.value = this.valueAccess(args.column.field, (args.data as
Object[]));
                (args.cell as CustomExcelCell).formula = args.value;
            }
        }
    }
    interface CustomExcelCell {
        formula: string;
    }
}

```

## MAIN.TS

```

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

```

## Limitations

- A CSV is a plain text format that does not support features such as cell rotation, font and color customization, column and row spanning, or adding formulas. CSV files store raw data without any formatting or styling.

## See Also

- [Exporting Grid in Cordova application](#)
- [How to get grid display numbers without grouping and same format to be exported to excel in Angular Grid](#)

## Global local in Angular Grid component

The Syncfusion Angular Grid component provides a feature known as Globalization (global and local), which makes the application more accessible and useful for individuals from different regions and language backgrounds. You have the ability to view data in your preferred language and format, resulting in an enhanced overall experience.

### Localization

The Syncfusion Angular Grid provides a built-in [Localization](#) library, enabling you to customize the text used in the grid to suit different languages or cultural preferences. With this library, you can change static text on various elements, such as **group drop area text** and **pager information text**, to different cultures, such as **Arabic**, **Deutsch**, **French**, and more.

This can be achieved by defining the [locale](#) property and translation object.

The following list of properties and its values are used in the grid.

### Data Rendering

Locale key words | Text | Example

Order ID	Verified
10248	true
10249	true
10250	true

True | true |



Order ID	Verified
10248	false
10249	false
10250	false

False | false |

Order ID	Customer Name
10248	VINET

Elements

Console

Sources


Network

Performance

```
<td class="e-rowcell" tabindex="-1" aria-label="10248 column header Order ID" data-colindex="0" aria-colindex="1" index="0">10248</td> == $0
```

ColumnHeader | column header |

TemplateCell | is template cell |

Employee Image	Employee ID
	1

Elements

Console

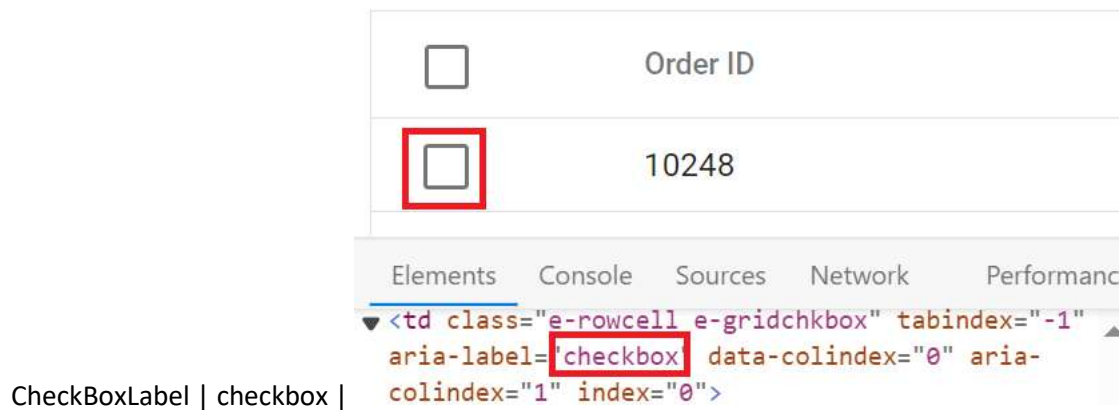
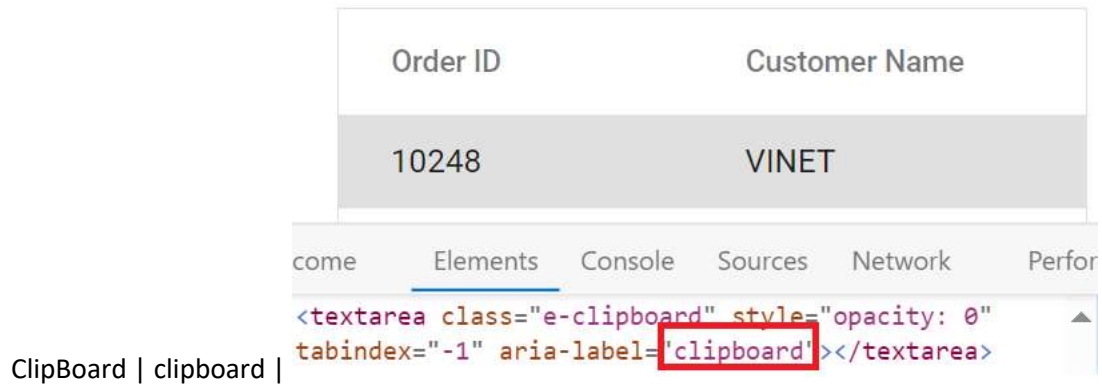
Sources

Network

Performance

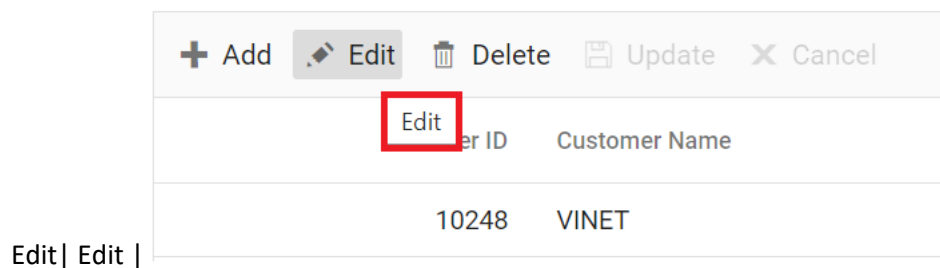
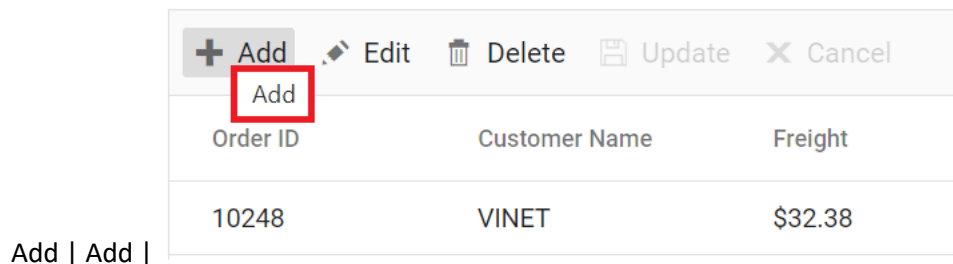
M

```
<td class="e-rowcell e-templatecell" tabindex="-1" aria-label="is template cell column header Employee Image" data-colindex="0" aria-colindex="1" index="0" style="text-align: center;">
```



### ColumnChooser

Locale key words | Text | Example



Cancel | Cancel |

+ Add   . Edit   🗑 Delete   💾 Update   ✕ Cancel	
Order ID	Customer Name
10248	VINET

Update | Update |

+ Add   . Edit   🗑 Delete   💾 Update   ✕ Cancel	
Order ID	Customer Name
10248	VINETY

Delete | Delete |

+ Add   . Edit   🗑 Delete   💾 Update   ✕ Cancel	
Order ID	Customer Name
10248	VINET

Save | Save |

💾   ✕
Save

EditOperationAlert | No records selected for edit operation |

No records selected for edit operation
OK

DeleteOperationAlert | No records selected for delete operation |

No records selected for delete operation
OK

SaveButton | Save |

SAVE	CANCEL
------	--------

OKButton | OK | **OK** CANCEL

CancelButton | Cancel | **SAVE** **CANCEL**

**Details of 10248** ×

Order ID

10248

EditFormTitle | Details of |

**Add New Record** ×

Order ID

AddFormTitle | Add New Record |

BatchSaveConfirm | Are you sure you want to save changes? |

**Are you sure you want to save changes?**

**OK** CANCEL

BatchSaveLostChanges | Unsaved changes will be lost. Are you sure you want to continue? |

**Unsaved changes will be lost. Are you sure you want to continue?**

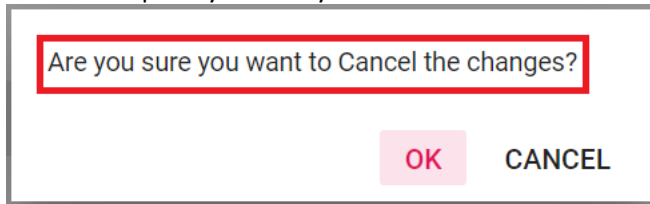
**OK** CANCEL

ConfirmDelete | Are you sure you want to Delete Record? |

**Are you sure you want to Delete Record?**

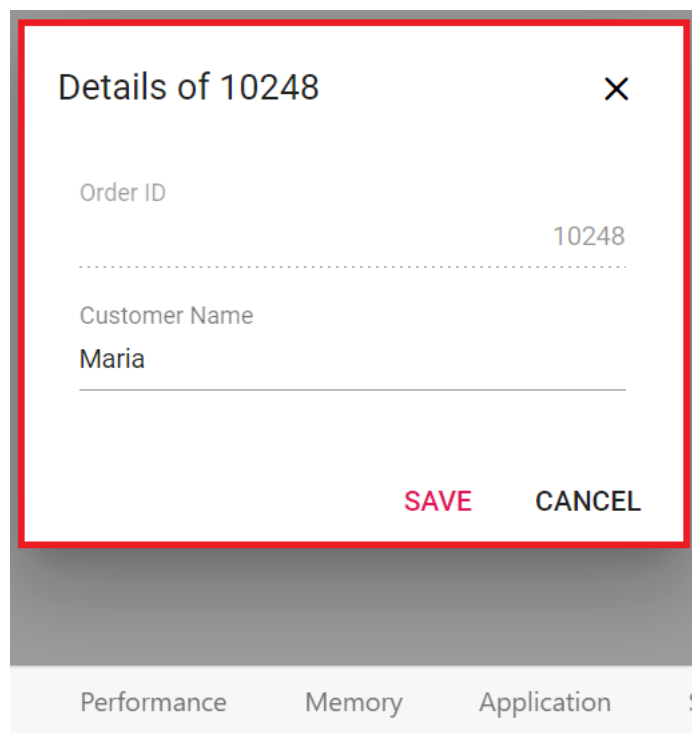
**OK** CANCEL

CancelEdit | Are you sure you want to Cancel the changes? |



DialogEditARIA | Edit dialog |

CommandColumnAria | is Command column column header |



```
width: 330px; max-height: 405px; z-index: 100000;  
;-c3990564242 aria-label="Dialog edit" class="e-
```

DialogEdit | Dialog edit |

## Grouping

Locale key words | Text | Example

InvalidFilterMessage | Invalid Filter Data

Order ID	Customer Name
Order ID's filter bar cell	

FilterbarTitle | \s filter bar cell |

▼ Clear Filter

Number Filters &gt;

103

×

No Matches Found

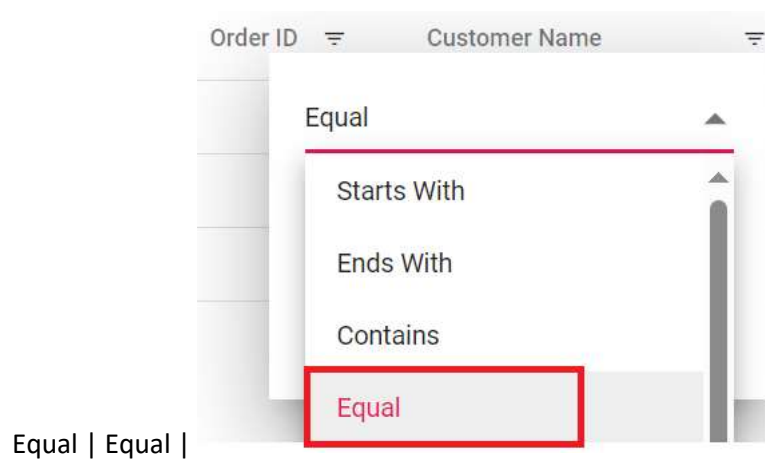
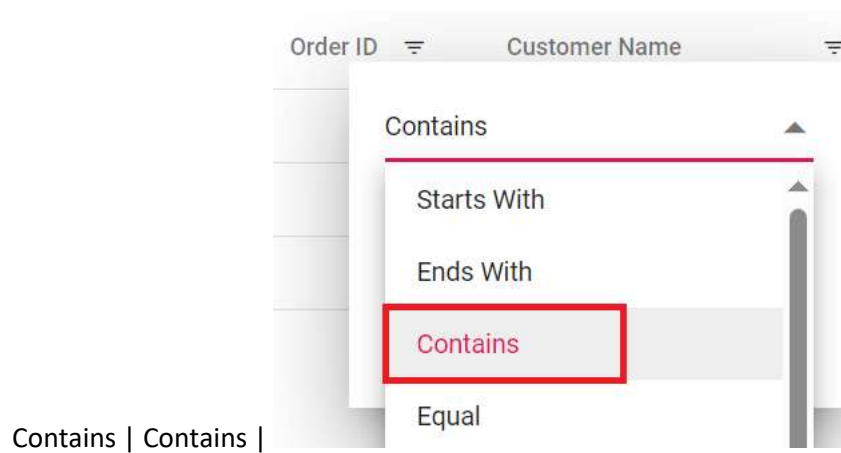
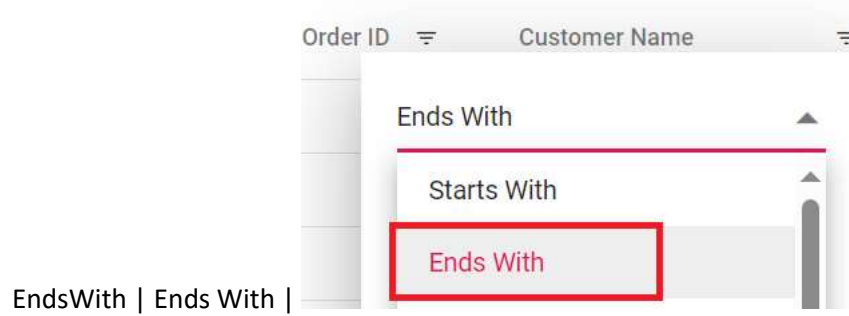
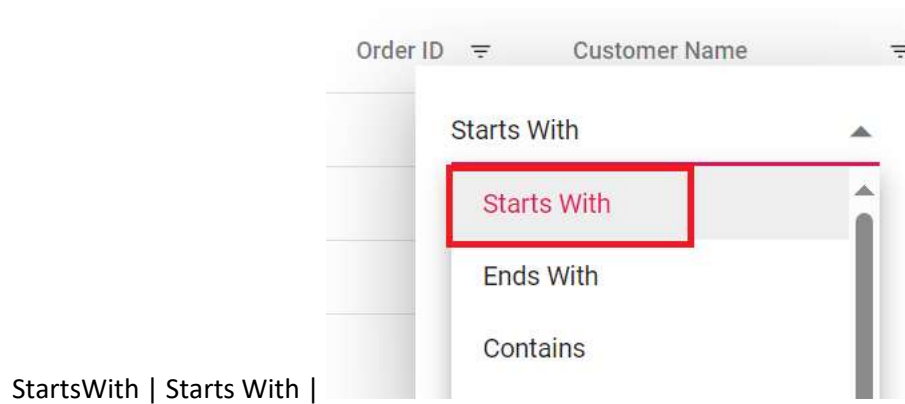
Matches | No Matches Found |

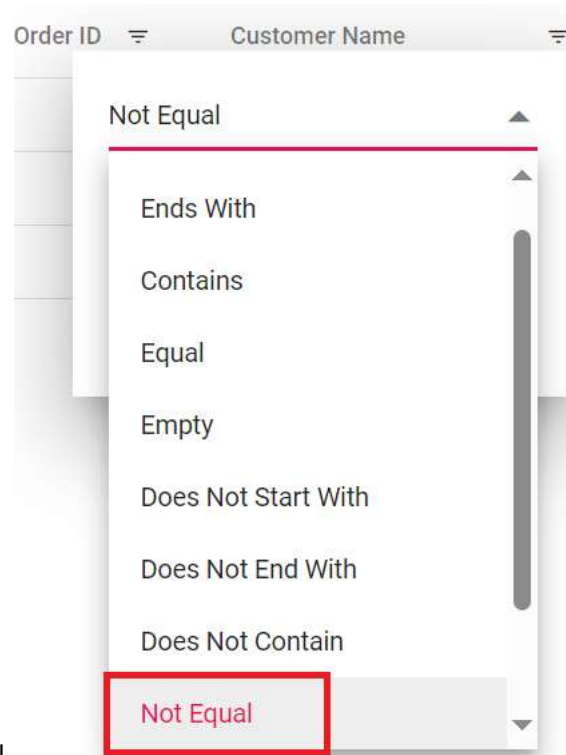
Order ID	Customer Name
Starts With	
VINET	
<b>FILTER</b>	CLEAR

FilterButton | Filter |

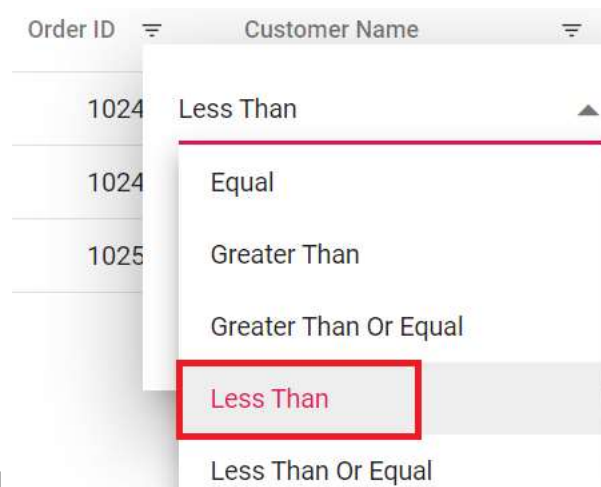
Order ID	Customer Name
Starts With	
VINET	
<b>FILTER</b>	<b>CLEAR</b>

ClearButton | Clear |



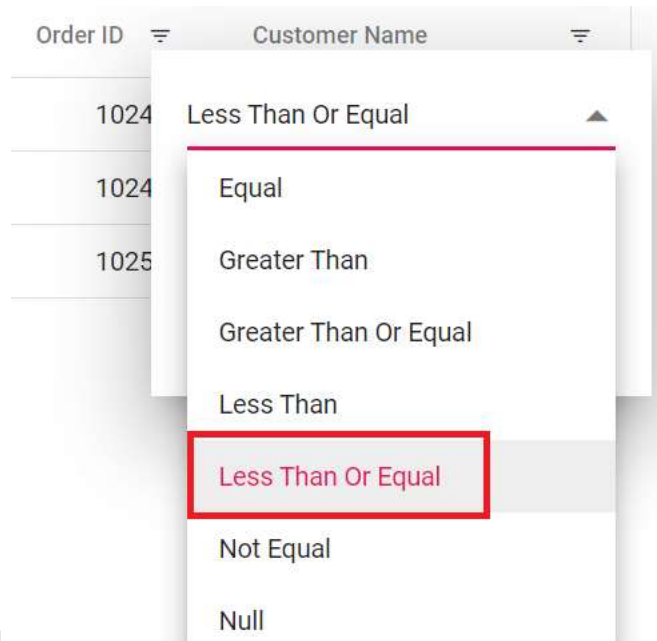


NotEqual | Not Equal |

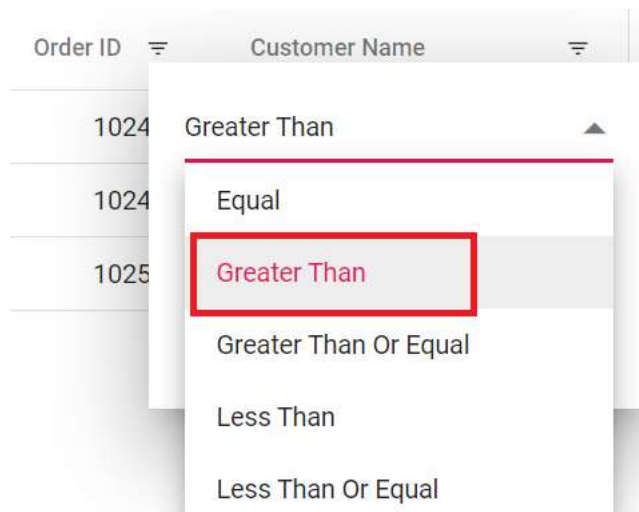


LessThan | Less Than |

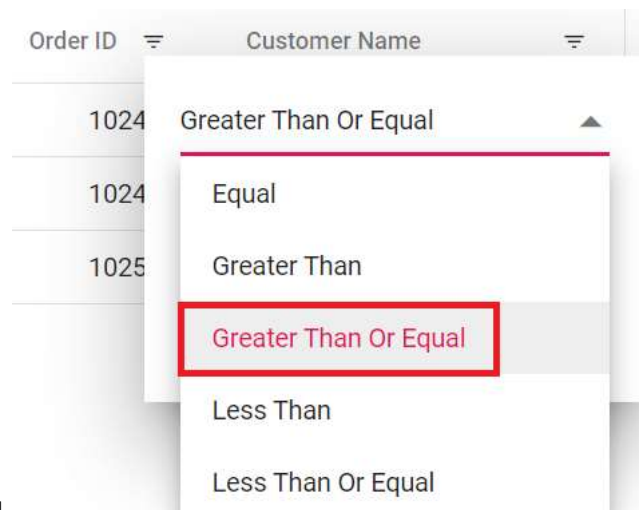




LessThanOrEqual | Less Than Or Equal |



GreaterThan | Greater Than |



GreaterThanOrEqual | Greater Than Or Equal |

ChooseDate | Choose a Date |

Order ID    Order Date

Equal

Choose a Date

FILTER    CLEAR

EnterValue | Enter the value |

Order ID    Customer Name

Starts With

Enter the value

FILTER    CLEAR

SelectAll | Select All |

Order ID    Customer Name

Clear Filter

Text Filters

Search

☒ Select All

Blanks | Blanks |

Order ID    Customer Name

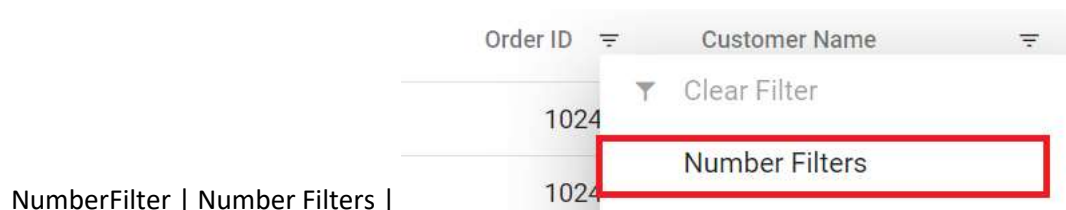
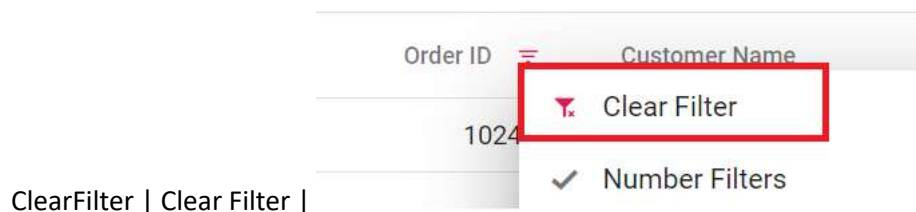
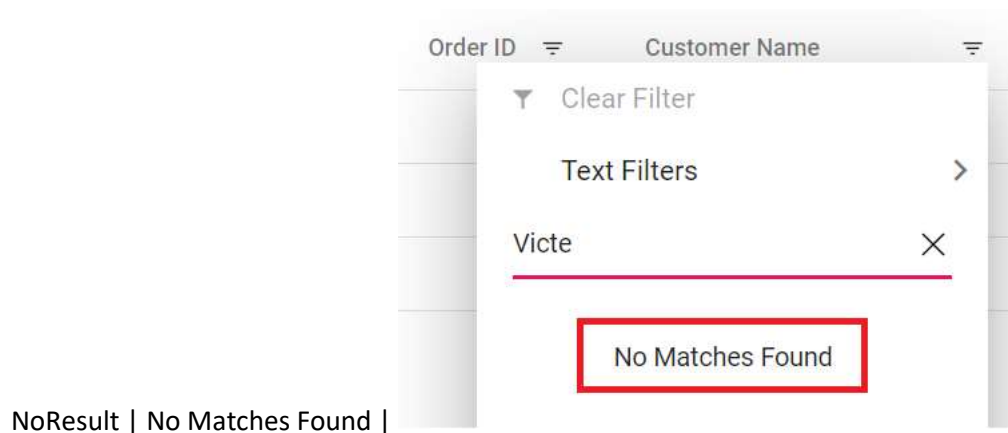
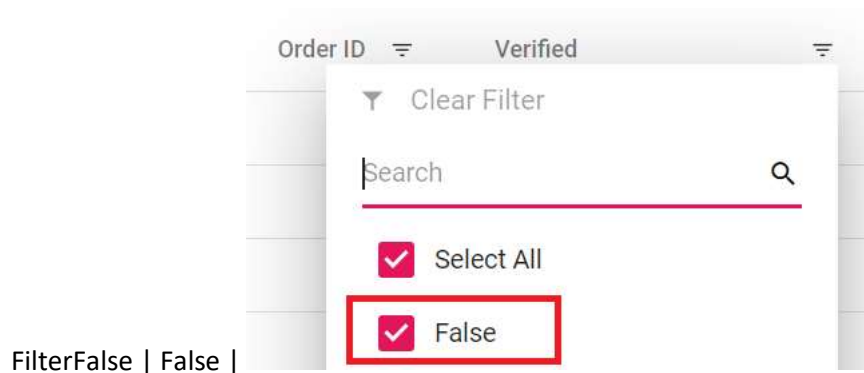
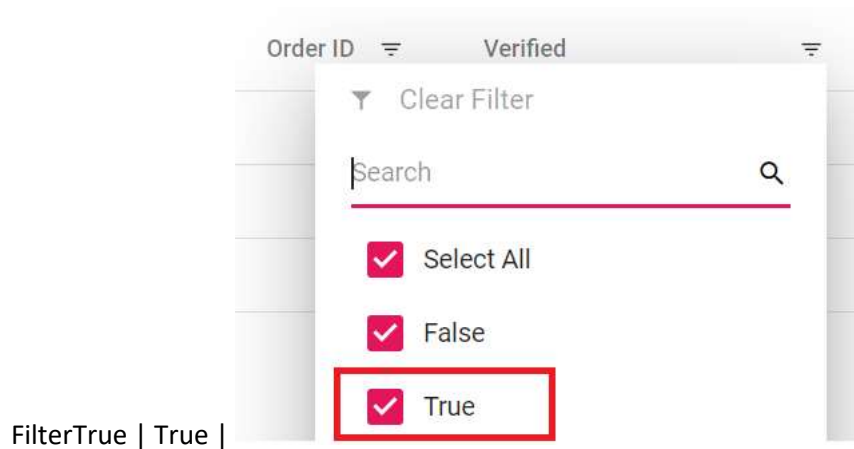
Clear Filter

Text Filters

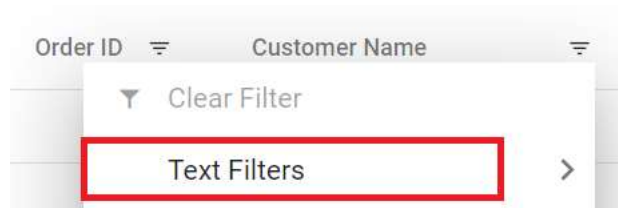
Search

☒ Select All

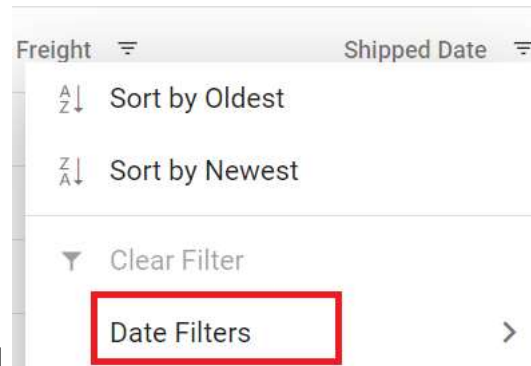
☒ Blanks



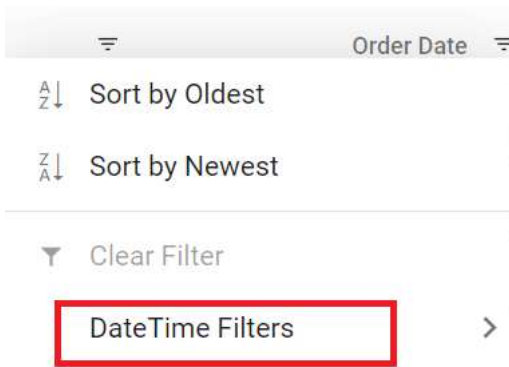
TextFilter | Text Filters |



DateFilter | Date Filters |



DateTimeFilter | DateTime Filters |



Custom Filter



Show rows where:

Customer Name

Starts With



Enter the value



AND

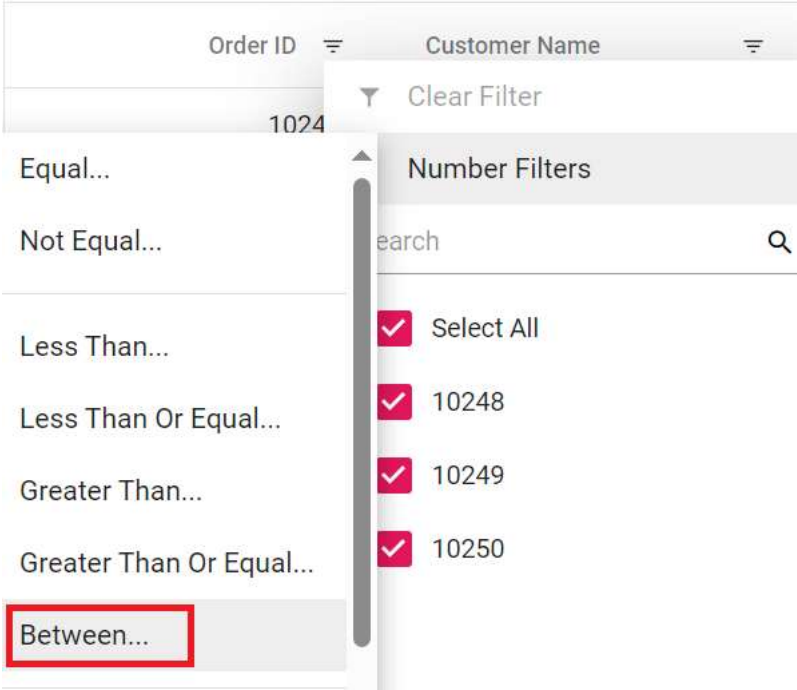


OR



Match Case

MatchCase | Match Case |



Order ID    Customer Name

1024

Clear Filter

Number Filters

Search

Equal...

Not Equal...

Less Than...

Less Than Or Equal...

Greater Than...

Greater Than Or Equal...

Between...

Between | Between |

Custom Filter

Show rows where:

Customer Name

Starts With    Enter the value

☒ AND   ☐ OR   ☐ Match Case

Enter the value

OK   CANCEL

CustomFilter | Custom Filter |

CustomFilterPlaceholder | Enter the value |

Custom Filter

Show rows where:

Customer Name

Starts With    Enter the value

CustomFilterDatePlaceholder | Choose a date |

Custom Filter



Show rows where:

Order Date

Equal



Choose a date



Custom Filter



Show rows where:

Customer Name

Starts With



Enter the value



AND



OR



Match Case

AND | AND |

Custom Filter



Show rows where:

Customer Name

Starts With



Enter the value



AND



OR



Match Case

OR | OR |

ShowRowsWhere | Show rows where: |

Custom Filter



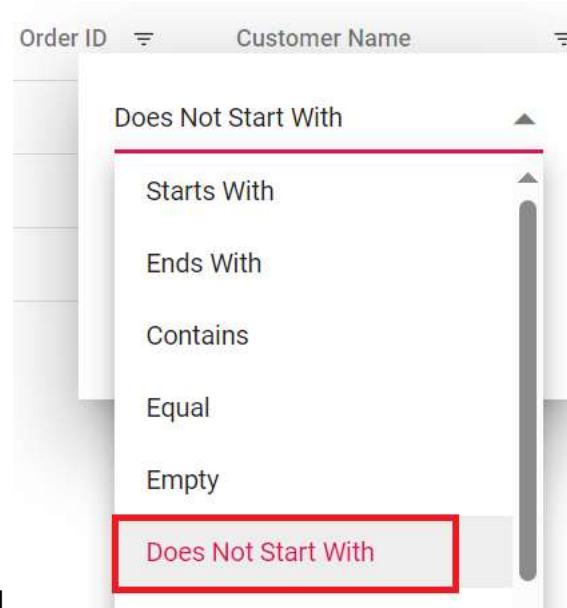
Show rows where:

Customer Name

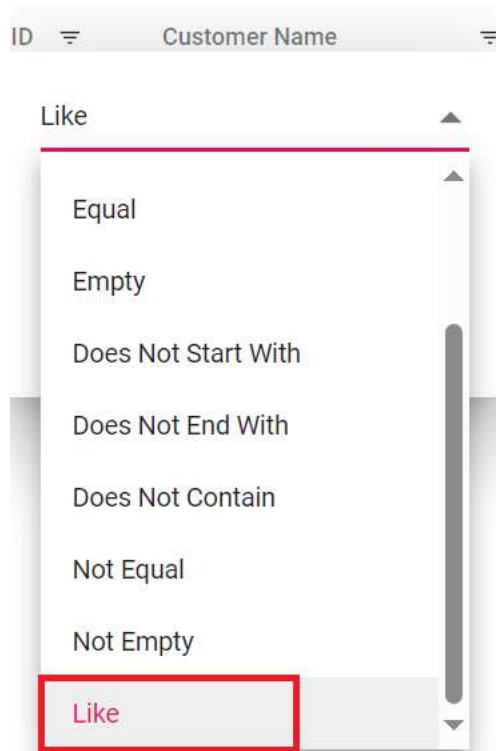
Starts With



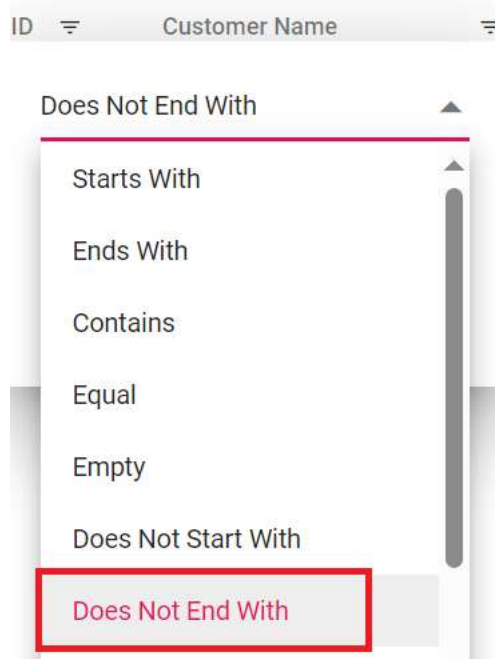
Enter the value



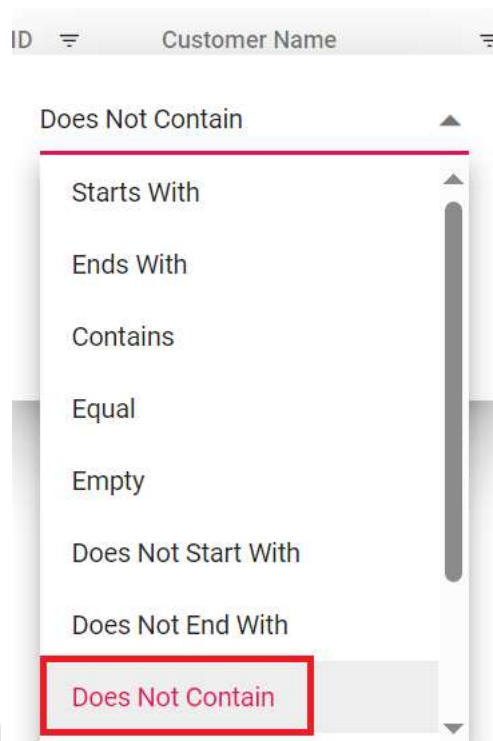
NotStartsWith | Does Not Start With |



Like | Like |



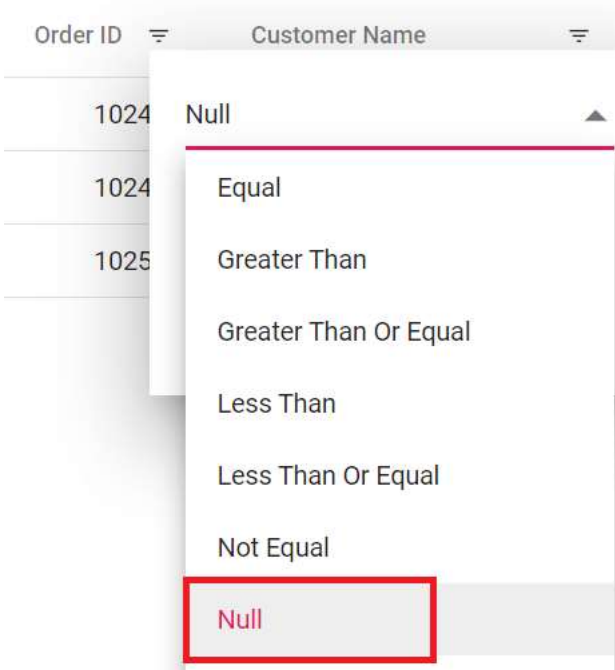
NotEndsWith | Does Not End With |



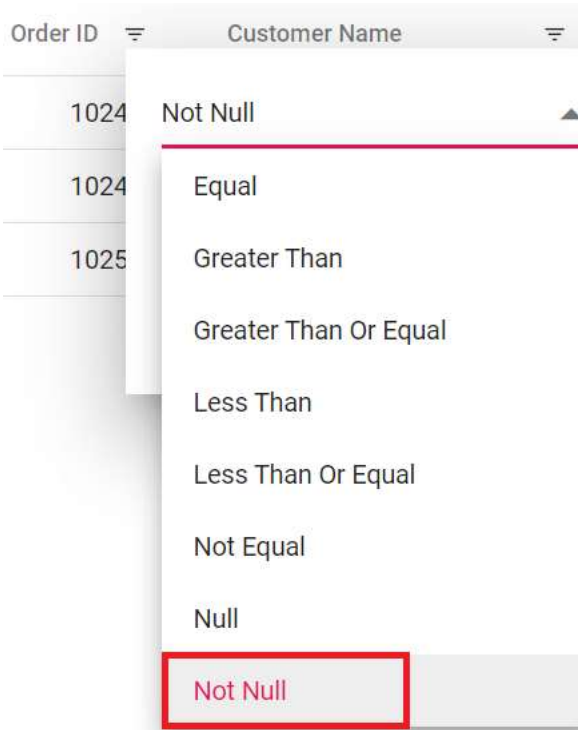
NotContains | Does Not Contain |



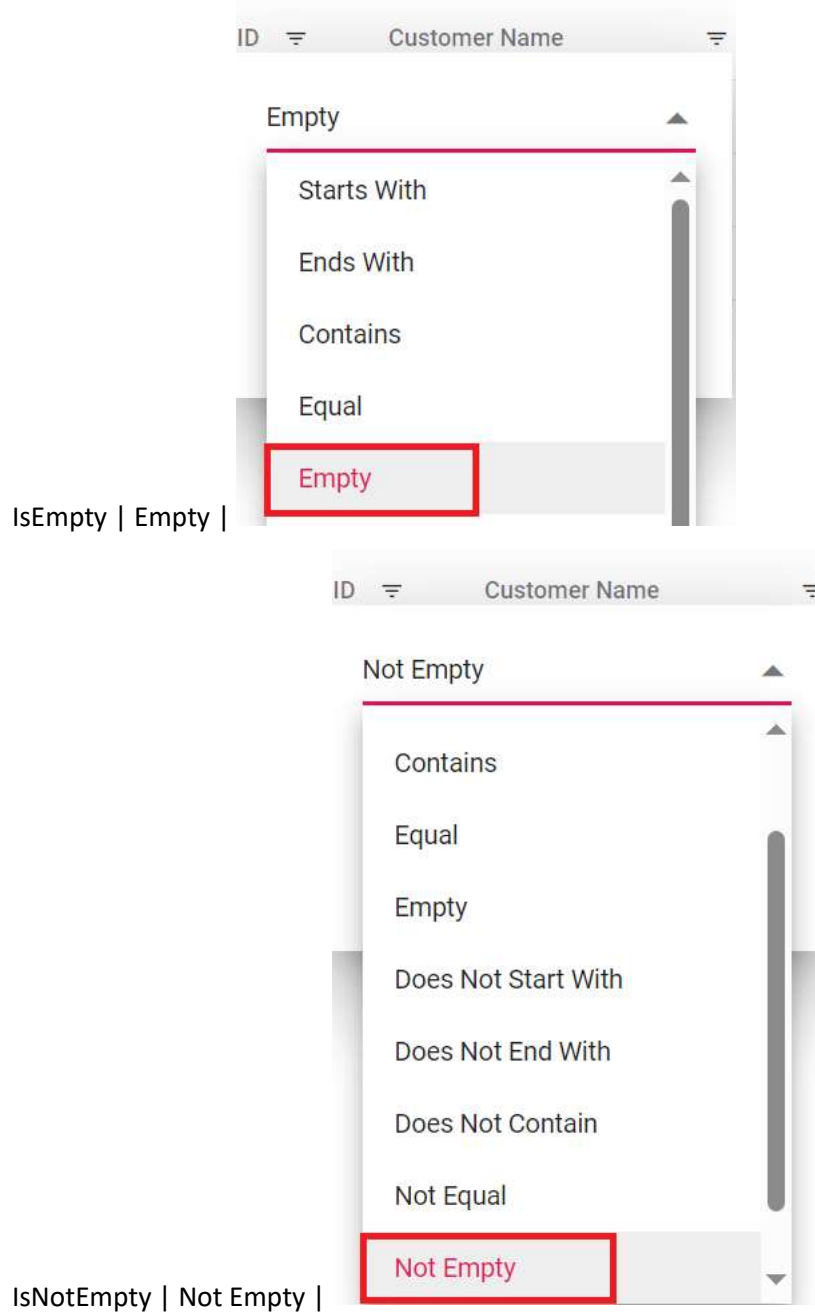
IsNull | Null |



Not Null | Not Null |



The image shows two screenshots of an Angular Grid component. The first screenshot shows a dropdown menu for the 'Customer Name' column with the 'Null' option selected and highlighted with a red box. The second screenshot shows the same dropdown menu with the 'Not Null' option selected and highlighted with a red box. The grid columns are 'Order ID' and 'Customer Name'. The data rows show 'Order ID' values 1024 and 1025, and 'Customer Name' values 'Null' and 'Not Null'.



Order ID    Customer Na...

Clear Filter

Text Filters >

vin

☒ Select All

☐ Add current selection to filter

☒ VINET

AddCurrentSelection | Add current selection to filter |

FilterMenuDialogARIA | Filter menu dialog |

Order ID	Customer Name
10248	
10249	
10250	
10251	

Equal

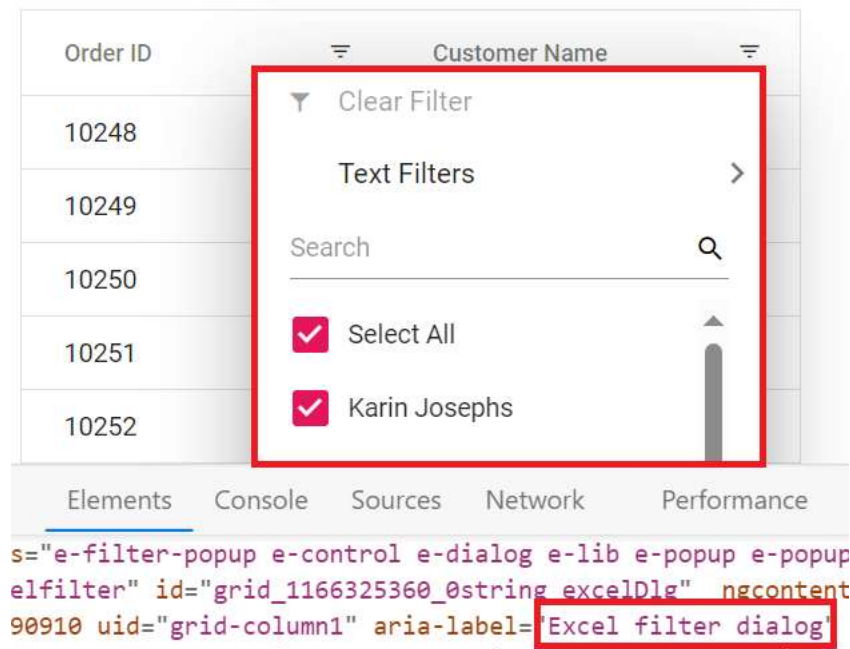
Enter the value

**FILTER** **CLEAR**

Elements Console Sources Network Performance

```
s="e-flmenu e-control e-dialog e-lib e-filter-popup e-po  
p-open" id="grid-column1709-flmdl" _ngcontent-ng-  
l0 aria-label="Filter menu dialog" role="dialog" style=
```

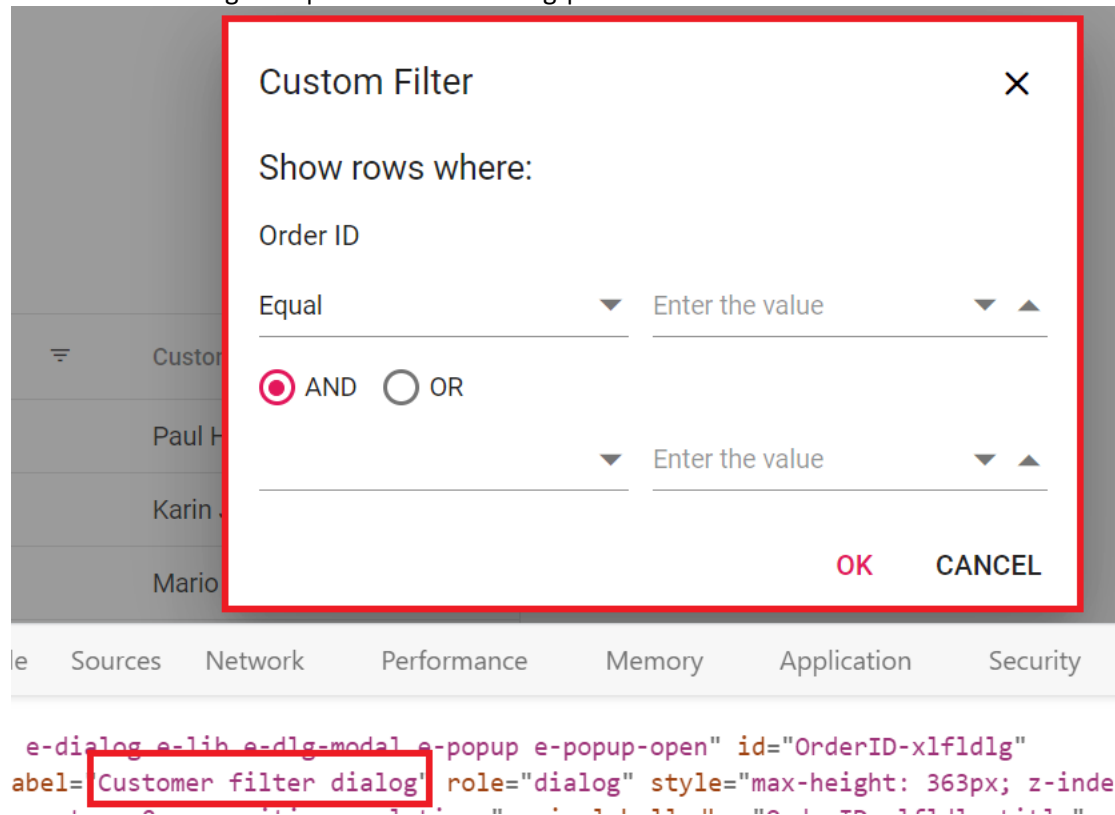
## ExcelFilterDialogARIA | Excel filter dialog |



The screenshot shows a data grid with columns 'Order ID' and 'Customer Name'. The 'Customer Name' column is filtered, and the 'Excel filter dialog' is open. The dialog has a 'Clear Filter' button, a 'Text Filters' section with a search bar, and two checked items: 'Select All' and 'Karin Josephs'. The dialog is highlighted with a red border.

```
s="e-filter-popup e-control e-dialog e-lib e-popup e-popup  
elfilter" id="grid_1166325360_0string_excelDlg" ngcontent  
90910 uid="grid-column1" aria-label="Excel filter dialog"
```

## CustomFilterDialogARIA | Custom filter dialog |

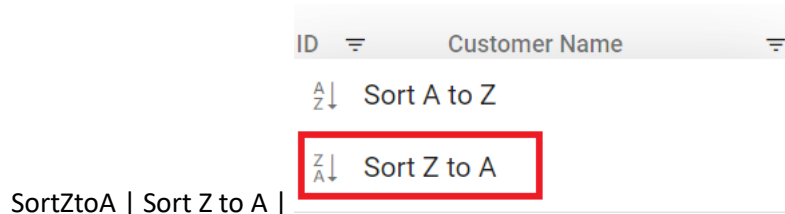


The screenshot shows a data grid with columns 'Order ID' and 'Customer Name'. The 'Customer Name' column is filtered, and the 'Custom filter dialog' is open. The dialog has a title 'Custom Filter', a close button, and a section 'Show rows where:'. It contains two filter rules for 'Order ID' with the operator 'Equal' and a search bar. The dialog is highlighted with a red border.

```
e-dialog e-lib e-dlg-modal e-popup e-popup-open" id="OrderID-xfldlg"  
abel="Customer filter dialog" role="dialog" style="max-height: 363px; z-inde
```



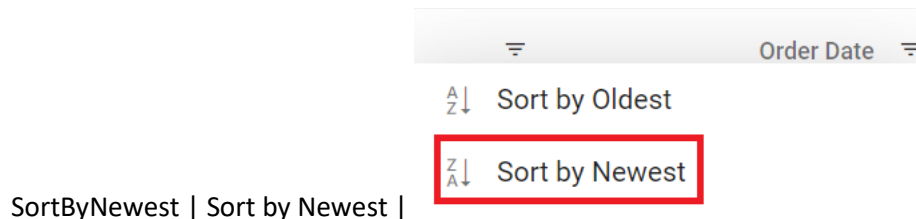
SortAtoZ | Sort A to Z |



SortZtoA | Sort Z to A |



SortByOldest | Sort by Oldest |



SortByNewest | Sort by Newest |



SortSmallestToLargest | Sort Smallest to Largest |



SortLargestToSmallest | Sort Largest to Smallest |

FilterDescription | Press Alt Down to open filter Menu |

Order ID	Customer Name
10248	Paul Henriot
10249	Karin Josephs
10250	Mario Pontes

Elements Console Sources Network Performance

`<th class="e-headercell e-filter-icon" _ngcontent-ng-c4169890910 tabindex="0" data-colindex="0" aria-colindex="1" aria-description="Press Alt Down to open filter Menu"`

### Searching

Locale key words | Text | Example

Sort | Sort | 

Sort

Order ID	Customer Name
10248	Paul Henriot
10249	Karin Josephs

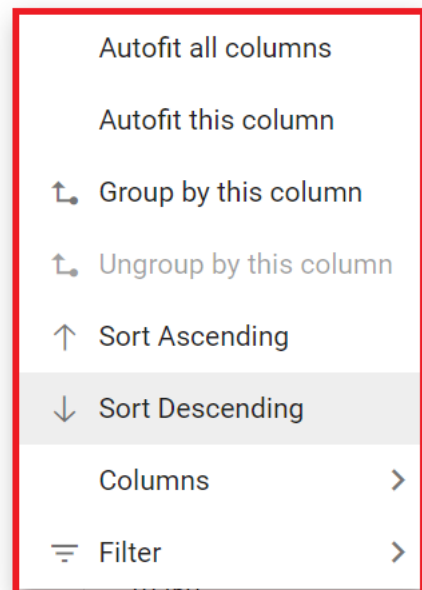
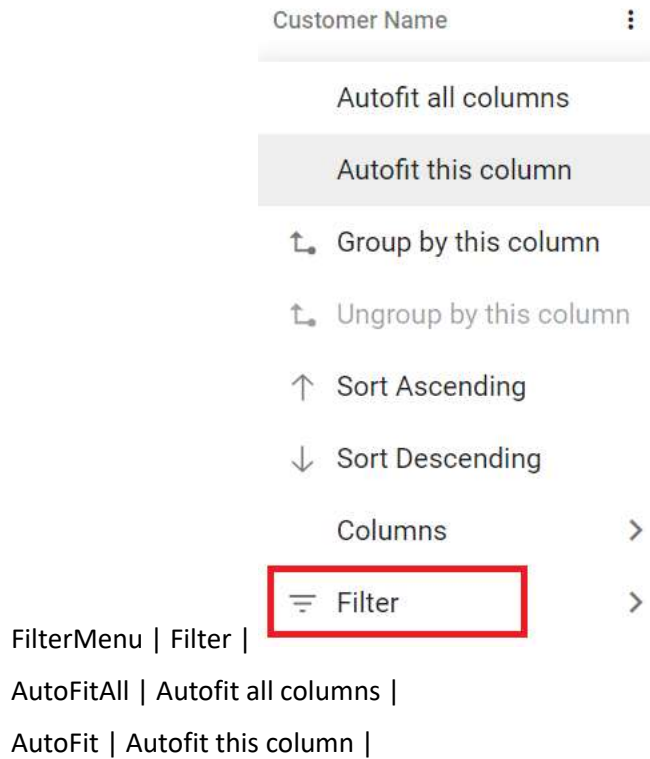
Elements Console Sources Network Performance

`<th class="e-headercell e-mousepointer e-focus" _ngcontent-ng-c3207075763 tabindex="0" data-colindex="0" aria-colindex="1" aria-sort="ascending" aria-description="Press Enter to sort" aria-rowspan="1" aria-colspan="1">...</th> == $0`

SortDescription | Press Enter to sort |

### Toolbar

Locale key words | Text | Example



ColumnMenuDialogARIA | Column menu dialog |

ColumnMenuDescription | Press Alt Down to open Column Menu |

Drag a column header here to group its column	
Order ID	Customer Name
10248	Maria
10249	Ana Trujillo
10250	Antonio Moreno

ie Elements Console Sources Network Perf

```
sort="none" aria-grabbed="false" aria-
description="Press Enter to sort. Press Ctrl spa
e to group. Press Alt Down to open Column Menu
aria-reverse="1" aria-colspan="1"
```

### ContextMenu

Locale key words | Text | Example

currentPageInfo | {0} of {1} pages | 1 of 1 pages (3 items)

totalItemsInfo | ({0} items) | 1 of 1 pages (3 items)

firstPageTooltip | Go to first page |

1< < ... 6 7 8

Go to first page

lastPageTooltip | Go to last page |

8 9 10 ... > >|

Go to last page

nextPageTooltip | Go to next page |

8 9 10 ... > >|

Go to next page

previousPageTooltip | Go to previous page |

1< < ... 6 7 8

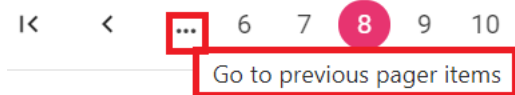
Go to previous page


nextPagerTooltip | Go to next pager items |


8 9 10 ... > >|


Go to next pager items





previousPagerTooltip | Go to previous pager items | 

pagerDropDown | Items per page | 

pagerAllDropDown | Items | 

All | All | 


totalItemInfo | ({0} item) | 

Container | Pager Container | 

Elements Console Sources Network Performance Memory Application S

```
ss="e-gridpager e-control e-pager e-lib" _ngcontent-ng-c1913795131 data-role="pager"
="-1" aria-label="Pager Container" >...</div> == $0
```

Information | Pager Information |

Order ID	Customer Na...
10248	
10248	Paul Henriot
	
Order ID: 10248	

Welcome Elements Console Sources Ne

```
<div class="e-pagerexternalmsg" aria-label="Pager external message" style>
Order ID: 10248</div> == $0
```

ExternalMsg | Pager external message |

Page | Page |

Of | of |  1 of 1 pages (5 items)

Pages | Pages |

### *Loading translations for de culture*

The Syncfusion Angular Grid component provides a built-in Localization library that allows you to load translation objects for different cultures. By using the **load** function of the **L10n** class, you can customize the text content of the Grid to be displayed in different languages.

This feature allows you to specify translation objects for specific cultures, such as **Deutsch** (German), and display the Grid's content in the desired language.

To work with **JSON** files in your application, you can enable JSON module resolution in TypeScript by adding the **resolveJsonModule** to true to your tsconfig.json file. Additionally, you can enhance module interoperation by setting **esModuleInterop** to true as shown below:

```
`ts
{
  compilerOptions: {
    resolveJsonModule: true,
    esModuleInterop: true,
  }
}
```

The following example demonstrates how to load a translation object for **Deutsch (de)** culture, by using the **load** function of **L10n** class from the **ej2-base** module and by defining the [locale](#) to **de-DE**.

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, PageService } from '@syncfusion/ej2-angular-grids'
import { L10n, } from '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel } from '@syncfusion/ej2-angular-grids';
import deDELocalization from './locale.json'
L10n.load(deDELocalization);
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [locale]='locale'
[allowGrouping]='true' [allowPaging]='true'
[pageSettings]='pageOptions' height='220px'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
```

```

        <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
</ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageOptions?: PageSettingsModel;
        public locale: any = "de-DE";
        ngOnInit(): void {
            this.data = data;
            this.pageOptions = { pageSize: 6 };
        }
    }
}

```

### LOCALE.JSON

```

{
    "de-DE": {
        "grid": {
            "EmptyRecord": "Keine Datensätze zum Anzeigen",
            "True": "wahr",
            "False": "falsch",
            "InvalidFilterMessage": "Ungültige Filterdaten",
            "GroupDropArea": "Ziehen Sie eine Spaltenüberschrift hierher, um die
Spalte zu gruppieren",
            "UnGroup": "Klicken Sie hier, um die Gruppierung aufzuheben",
            "GroupDisable": "Die Gruppierung ist für diese Spalte deaktiviert",
            "FilterbarTitle": "\"s Filterbalkenzelle",
            "EmptyDataSourceError": "Datenquelle darf beim ersten Laden nicht
leer sein, da Spalten aus der Datenquelle automatisch generiert werden",
            "Add": "Hinzufügen",
            "Edit": "Bearbeiten",
            "Cancel": "Abbrechen",
            "Update": "Aktualisieren",
            "Delete": "Löschen",
            "Print": "Drucken",
            "Pdfexport": "PDF-Export",
            "Excelexport": "Excel-Export",
            "Wordexport": "Word-Export",
            "Csvexport": "CSV-Export",
            "Search": "Suchen",
            "Columnchooser": "Spalten",
            "Save": "Speichern",
            "Item": "Datensatz",
            "Items": "Datensätze",
            "EditOperationAlert": "Keine Datensätze zum Bearbeiten ausgewählt",
            "DeleteOperationAlert": "Keine Datensätze zum Löschen ausgewählt",
            "SaveButton": "Speichern",
            "OKButton": "in Ordnung",
            "CancelButton": "Abbrechen",
            "EditFormTitle": "Details von",

```

```

    "AddFormTitle": "Neuen Datensatz hinzufügen",
    "BatchSaveConfirm": "Möchten Sie die Änderungen wirklich
speichern?",
    "BatchSaveLostChanges": "Nicht gespeicherte Änderungen gehen
verloren. Sind Sie sicher, dass Sie fortfahren wollen?",
    "ConfirmDelete": "Möchten Sie den Datensatz wirklich löschen?",
    "CancelEdit": "Möchten Sie die Änderungen wirklich abbrechen?",
    "ChooseColumns": "Wählen Sie Spalten",
    "SearchColumns": "Spalten durchsuchen",
    "Matches": "Keine Treffer gefunden",
    "FilterButton": "Filter",
    "ClearButton": "Löschen",
    "StartsWith": "Beginnt mit",
    "EndsWith": "Endet mit",
    "Contains": "Enthält",
    "Equal": "Gleich",
    "NotEqual": "Nicht gleich",
    "LessThan": "Weniger als",
    "LessThanOrEqual": "Weniger als oder gleich",
    "GreaterThan": "Grösser als",
    "GreaterThanOrEqual": "Grösser als oder gleich",
    "ChooseDate": "Wählen Sie ein Datum",
    "EnterValue": "Geben Sie den Wert ein",
    "Copy": "Kopieren",
    "Group": "Nach dieser Spalte gruppieren",
    "Ungroup": "Gruppierung nach dieser Spalte aufheben",
    "autoFitAll": "Automatisch alle Spalten anpassen",
    "autoFit": "Diese Spalte automatisch anpassen",
    "Export": "Export",
    "FirstPage": "Erste Seite",
    "LastPage": "Letzte Seite",
    "PreviousPage": "Vorherige Seite",
    "NextPage": "Nächste Seite",
    "SortAscending": "Aufsteigend sortieren",
    "SortDescending": "Absteigend sortieren",
    "EditRecord": "Datensatz bearbeiten",
    "DeleteRecord": "Datensatz löschen",
    "FilterMenu": "Filter",
    "SelectAll": "Wählen Sie Alle",
    "Blanks": "Leerzeichen",
    "FilterTrue": "Wahr",
    "FilterFalse": "Falsch",
    "NoResult": "Keine Treffer gefunden",
    "ClearFilter": "Filter löschen",
    "NumberFilter": "Anzahl Filter",
    "TextFilter": "Textfilter",
    "DateFilter": "Datumsfilter",
    "DateTimeFilter": "DatumsZeit-Filter",
    "MatchCase": "Gross- / Kleinschreibung",
    "Between": "Zwischen",
    "CustomFilter": "Benutzerdefinierte Filter",
    "CustomFilterPlaceholder": "Geben Sie einen Wert ein",
    "CustomFilterDatePlaceholder": "Wählen Sie ein Datum",
    "AND": "UND",
    "OR": "ODER",
    "ShowRowsWhere": "Zeilen anzeigen, in denen:",
    "NotStartsWith": "beginnt nicht mit",

```

```

    "Like": "Wie",
    "NotEndsWith": "endet nicht mit",
    "NotContains": "enthält nicht",
    "IsNull": "Null",
    "NotNull": "nicht null",
    "IsEmpty": "leer",
    "IsNotEmpty": "nicht leer",
    "AddCurrentSelection": "Aktuelle Auswahl zum Filter hinzufügen",
    "UnGroupButton": "Klicken Sie hier, um die Gruppierung aufzuheben",
    "AutoFitAll": "Automatisch alle Spalten anpassen",
    "AutoFit": "Diese Spalte automatisch anpassen",
    "Clear": "Klar",
    "FilterMenuDialogARIA": "Menüdialog filtern",
    "ExcelFilterDialogARIA": "Excel-Filterdialog",
    "DialogEditARIA": "Dialog bearbeiten",
    "ColumnChooserDialogARIA": "Spaltenauswahl",
    "ColumnMenuDialogARIA": "Spaltenmenüdialog",
    "CustomFilterDialogARIA": "Benutzerdefinierter Filterdialog",
    "SortAtoZ": "Sortiere von A bis Z",
    "SortZtoA": "Z bis A sortieren",
    "SortByOldest": "Nach Ältesten sortieren",
    "SortByNewest": "Nach Neuesten sortieren",
    "SortSmallestToLargest": "Vom Kleinsten zum Größten sortieren",
    "SortLargestToSmallest": "Vom Größten zum Kleinsten sortieren",
    "Sort": "Sortieren",
    "FilterDescription": "Drücken Sie die Alt-Nach-unten-Taste, um das
Filtermenü zu öffnen",
    "SortDescription": "Drücken Sie zum Sortieren die Eingabetaste",
    "ColumnMenuDescription": "Drücken Sie die Alt-Nach-unten-Taste, um
das Spaltenmenü zu öffnen",
    "GroupDescription": "Drücken Sie zum Gruppieren die Strg-Leertaste",
    "ColumnHeader": " Spaltenüberschrift ",
    "TemplateCell": " ist eine Vorlagenzelle",
    "CommandColumnAria": "ist die Spaltenüberschrift der Befehlsspalte
",
    "DialogEdit": "Dialog bearbeiten",
    "Clipboard": "Zwischenablage",
    "GroupButton": "Gruppenschaltfläche",
    "UnGroupAria": "Schaltfläche zum Aufheben der Gruppierung",
    "GroupSeperator": "Trennzeichen für die gruppierten Spalten",
    "UnGroupIcon": "heben Sie die Gruppierung der gruppierten Spalte auf
",
    "GroupedSortIcon": "sortieren Sie die gruppierte Spalte ",
    "GroupedDrag": "Ziehen Sie die gruppierte Spalte",
    "GroupCaption": " ist eine Gruppenunterschriftszelle",
    "CheckBoxLabel": "Kontrollkästchen",
    "Expanded": "Erweitert",
    "Collapsed": "Zusammengebrochen"
  },
  "pager": {
    "currentPageInfo": "{0} von {1} Seiten",
    "totalItemsInfo": "({0} Datensätze)",
    "firstPageTooltip": "Gehe zur ersten Seite",
    "lastPageTooltip": "Gehe zur letzten Seite",
    "nextPageTooltip": "Gehe zur nächsten Seite",
    "previousPageTooltip": "Zurück zur letzten Seite",
    "nextPagerTooltip": "Gehe zu den nächsten Pager-Elementen",
  }

```

```

        "previousPagerTooltip": "Gehen Sie zu den vorherigen Pager-
Elementen",
        "pagerDropDown": "Datensätze pro Seite",
        "pagerAllDropDown": "Datensätze",
        "All": "Alle",
        "totalItemInfo": "({0} Datensatz)",
        "Container": "Pager-Container",
        "Information": "Pager-Informationen",
        "ExternalMsg": "Pager-externe Nachricht",
        "Page": "Buchseite ",
        "Of": " von ",
        "Pages": " Seiten"
    }
}
}

```

### MAIN.TS

```

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

```

#### Loading translations for fr culture

The Loading translations feature allows you to specify translation objects for different cultures, such as **Deutsch**, **Arabic**, **French** and display the Grid's content in the desired language.

To work with **JSON** files in your application, you can enable JSON module resolution in TypeScript by adding the **resolveJsonModule** to true to your tsconfig.json file. Additionally, you can enhance module interoperability by setting **esModuleInterop** to true as shown below:

```

`ts
{
  compilerOptions: {
    resolveJsonModule: true,
    esModuleInterop: true,
  }
}
`

```

The following example demonstrates how to load a translation object for **French (fr)** culture, by defining the [locale](#) to **fr-FR** and by using the **load** function of **L10n** class from the **ej2-base** module.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { GridModule, GroupService, PageService, EditService, ToolbarService } from '@syncfusion/ej2-angular-grids';
import { L10n } from '@syncfusion/ej2-base';

```

```

import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel, EditSettingsModel, ToolbarItems } from
 '@syncfusion/ej2-angular-grids';
import frFRLocalization from './locale.json';
L10n.load(frFRLocalization);
@Component({
  imports: [

    GridModule
  ],
  providers: [GroupService, PageService, EditService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [locale]='locale'
[allowGrouping]='true' [allowPaging]='true'
    [pageSettings]='pageOptions' [editSettings]='editSettings'
[toolbar]='toolbar' height='220px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public pageOptions?: PageSettingsModel;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public locale: any = 'fr-FR';
  ngOnInit(): void {
    this.data = data;
    this.pageOptions = { pageSize: 6 };
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
'Search'];
  }
}

```

## LOCALE.JSON

```

{
  "fr-FR": {
    "grid": {
      "EmptyRecord": "Aucun enregistrement à afficher",
      "True": "vrai",
      "False": "faux",
      "InvalidFilterMessage": "Données de filtre non valides",

```

```

    "GroupDropArea": "Faites glisser un en-tête de colonne ici pour
regrouper sa colonne",
    "UnGroup": "Cliquez ici pour dissocier",
    "GroupDisable": "Le regroupement est désactivé pour cette colonne",
    "FilterbarTitle": "Cellule de barre de filtre \"s\",
    "EmptyDataSourceError": "DataSource ne doit pas être vide lors du
chargement initial car les colonnes sont générées à partir de dataSource
dans AutoGenerate Column Grid",
    "Add": "Ajouter",
    "Edit": "Éditer",
    "Cancel": "Annuler",
    "Update": "Mise à jour",
    "Delete": "Supprimer",
    "Print": "Impression",
    "Pdfexport": "Exportation PDF",
    "Excelexport": "Exportation Excel",
    "Wordexport": "Exportation de mots",
    "Csvexport": "Exportation CSV",
    "Search": "Chercher",
    "Columnchooser": "Colonnes",
    "Save": "sauvegarder",
    "Item": "article",
    "Items": "articles",
    "EditOperationAlert": "Aucun enregistrement sélectionné pour
l'opération d'édition",
    "DeleteOperationAlert": "Aucun enregistrement sélectionné pour
l'opération de suppression",
    "SaveButton": "Sauvegarder",
    "OKButton": "Ok",
    "CancelButton": "Annuler",
    "EditFormTitle": "Les détails de",
    "AddFormTitle": "Ajouter un nouvel enregistrement",
    "BatchSaveConfirm": "Voulez-vous vraiment enregistrer les
modifications ?",
    "BatchSaveLostChanges": "Les modifications non enregistrées seront
perdues. Es-tu sur de vouloir continuer ?",
    "ConfirmDelete": "Voulez-vous vraiment supprimer l'enregistrement ?",
    "CancelEdit": "Voulez-vous vraiment annuler les modifications ?",
    "ChooseColumns": "Choisissez la colonne",
    "SearchColumns": "colonnes de recherche",
    "Matches": "Aucun résultat",
    "FilterButton": "Filtrer",
    "ClearButton": "Effacer",
    "StartsWith": "Commence par",
    "EndsWith": "Se termine par",
    "Contains": "Contient",
    "Equal": "Égal",
    "NotEqual": "Différent",
    "LessThan": "Inférieur",
    "LessThanOrEqual": "Inférieur ou égal",
    "GreaterThan": "Supérieur",
    "GreaterThanOrEqual": "Supérieur ou égal",
    "ChooseDate": "Choisissez une date",
    "EnterValue": "Entrez la valeur",
    "Copy": "Copie",
    "Group": "Regrouper par cette colonne",
    "Ungroup": "Dissocier par cette colonne",

```



```

"autoFitAll": "Ajuster automatiquement toutes les colonnes",
"autoFit": "Ajuster automatiquement cette colonne",
"Export": "Exportation",
"FirstPage": "Première page",
"LastPage": "Dernière page",
"PreviousPage": "Page précédente",
"NextPage": "Page suivante",
"SortAscending": "Trier par ordre croissant",
"SortDescending": "Trier par ordre décroissant",
"EditRecord": "Modifier l'enregistrement",
"DeleteRecord": "Supprimer l'enregistrement",
"FilterMenu": "Filtre",
"SelectAll": "Tout sélectionner",
"Blanks": "Blancs",
"FilterTrue": "Vrai",
"FilterFalse": "Faux",
"NoResult": "Aucun résultat",
"ClearFilter": "Effacer le filtre",
"NumberFilter": "Filtres numériques",
"TextFilter": "Filtres de texte",
"DateFilter": "Filtres de date",
"DateTimeFilter": "Filtres DateTime",
"MatchCase": "Cas de correspondance",
"Between": "Entre",
"CustomFilter": "Filtre personnalisé",
"CustomFilterPlaceholder": "Entrez la valeur",
"CustomFilterDatePlaceholder": "Choisissez une date",
"AND": "ET",
"OR": "OU",
"ShowRowsWhere": "Afficher les lignes où:",
"NotStartsWith": "Ne commence pas par",
"Like": "Comme",
"NotEndsWith": "Ne se termine pas par",
"NotContains": "Ne contient pas",
"IsNull": "Nul",
"NotNull": "Non nul",
"IsEmpty": "Vide",
"IsNotEmpty": "Pas vide",
"AddCurrentSelection": "Ajouter la sélection actuelle au filtre",
"UnGroupButton": "Cliquez ici pour dissocier",
"AutoFitAll": "Ajuster automatiquement toutes les colonnes",
"AutoFit": "Ajuster automatiquement cette colonne",
"Clear": "Dégager",
"FilterMenuDialogARIA": "Boîte de dialogue du menu Filtre",
"ExcelFilterDialogARIA": "Boîte de dialogue de filtre Excel",
"DialogEditARIA": "Boîte de dialogue Modifier",
"ColumnChooserDialogARIA": "Sélecteur de colonne",
"ColumnMenuDialogARIA": "Boîte de dialogue du menu des colonnes",
"CustomFilterDialogARIA": "Boîte de dialogue Filtre personnalisé",
"SortAtoZ": "Trier de A à Z",
"SortZtoA": "Trier de Z à A",
"SortByOldest": "Trier par le plus ancien",
"SortByNewest": "Trier par Plus récent",
"SortSmallestToLargest": "Trier du plus petit au plus grand",
"SortLargestToSmallest": "Trier du plus grand au plus petit",
"Sort": "Sorte",

```

```

    "FilterDescription": "Appuyez sur Alt Bas pour ouvrir le menu du
    filtre",
    "SortDescription": "Appuyez sur Entrée pour trier",
    "ColumnMenuDescription": "Appuyez sur Alt Bas pour ouvrir le menu des
    colonnes",
    "GroupDescription": "Appuyez sur Ctrl espace pour grouper",
    "ColumnHeader": " en-tête de colonne ",
    "TemplateCell": " est une cellule modèle",
    "CommandColumnAria": "est l'en-tête de colonne de la colonne de
    commande ",
    "DialogEdit": "Modifier la boîte de dialogue",
    "Clipboard": "presse-papiers",
    "GroupButton": "Bouton de groupe",
    "UnGroupAria": "bouton dissocier",
    "GroupSeperator": "Séparateur pour les colonnes groupées",
    "UnGroupIcon": "dissocier la colonne groupée ",
    "GroupedSortIcon": "trier la colonne groupée ",
    "GroupedDrag": "Faites glisser la colonne groupée",
    "GroupCaption": " est une cellule de légende de groupe",
    "CheckBoxLabel": "case à cocher",
    "Expanded": "Étendue",
    "Collapsed": "S'est effondré"
  },
  "pager": {
    "currentPageInfo": "{0} de {1} pages",
    "totalItemsInfo": "({0} éléments)",
    "firstPageTooltip": "Aller à la première page",
    "lastPageTooltip": "Aller à la dernière page",
    "nextPageTooltip": "Aller à la page suivante",
    "previousPageTooltip": "Aller à la page précédente",
    "nextPagerTooltip": "Aller aux éléments suivants du téléavertisseur",
    "previousPagerTooltip": "Accéder aux éléments de téléavertisseur
    précédents",
    "pagerDropDown": "objets par page",
    "pagerAllDropDown": "Articles",
    "All": "Tout",
    "totalItemInfo": "({0} élément)",
    "Container": "Conteneur de téléavertisseur",
    "Information": "Informations sur le téléavertisseur",
    "ExternalMsg": "Message externe du téléavertisseur",
    "Page": "Page",
    "Of": " de ",
    "Pages": " pages"
  }
}
}
}

```

## 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 different localization

The Syncfusion Angular Grid allows you to switch the localization from one culture to another culture. This will be useful when you want to change the localization based on your requirements.

To switch to a different localization, follow these steps:

**Step 1:** Import and load the required CLDR (Common Locale Data Repository) data for the desired culture using the `loadCldr` function.

```
`ts
loadCldr(
  cagregorian,
  currencies,
  numbers,
  timeZoneNames,
  numberingSystems
);
`
```

**Step 2:** To import `json` files in your application, you can enable JSON module resolution in TypeScript by adding the `resolveJsonModule` to `true` to your `tsconfig.json` file. Additionally, you can enhance module interoperation by setting `esModuleInterop` to `true` as shown below:

```
`ts
{
  compilerOptions: {
    resolveJsonModule: true,
    esModuleInterop: true,
  }
}
`
```

**Step 3:** To change the default culture and the currency code, you can use the methods `setCulture` for setting the locale and `setCurrencyCode` for setting the currency code.

To switch to the **French** culture and set the currency code as **EUR**, you can use the `setCulture` method and the `setCurrencyCode` method of the Grid on external button click. This is demonstrated below:

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, ToolbarService,
EditService, ExcelExportService, PdfExportService, SearchService } from
'@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { L10n, loadCldr, setCulture, setCurrencyCode } from
'@syncfusion/ej2-base';
```

```

import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import cagregorian from './ca-gregorian.json';
import currencies from './currencies.json';
import numbers from './numbers.json';
import timeZoneNames from './timeZoneNames.json';
import numberingSystems from './numberingSystems.json';
import frFRLocalization from './locale.json';
L10n.load(frFRLocalization);
setCulture('fr-FR'); // Change the Grid culture
setCurrencyCode('EUR');
@Component({
  imports: [

    GridModule,
    ButtonModule
  ],
  providers: [PageService, ToolbarService,
  EditService, ExcelExportService, PdfExportService, SearchService ],
  standalone: true,
  selector: 'app-root',
  template: `
    <button ej-button id="frButton" cssClass="e-outline"
    (click)="ChangeFrLocale()">Change FR Locale</button>
    <button ej-button id="enButton" style="margin-left:5px" cssClass="e-
    outline" (click)="ChangeEnLocale()">Change En Locale</button>
    <ejs-grid style="padding: 10px 10px" [dataSource]='data'
    [allowPaging]='true' height='220px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
        textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
        width=100></e-column>
        <e-column field='Freight' headerText='Freight' format="C2"
        width=90></e-column>
        <e-column field="ShipCountry" headerText="Ship Country"
        width="100"></e-column>
      </e-columns>
    </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    ngOnInit(): void {
      this.data = data;
      loadCldr(
        cagregorian,
        currencies,
        numbers,
        timeZoneNames,
        numberingSystems
      );
    }
    ChangeFrLocale() {
      setCulture('fr-FR'); // Change the Grid culture to French locale
      setCurrencyCode('EUR'); // Change the currency code based on French
      culture
    }
  }

```

```

ChangeEnLocale() {
    setCulture('en-US'); // Change the Grid culture to English locale
    setCurrencyCode('USD'); // Change the currency code based on
    American English culture
}

```

## LOCALE.JSON

```

{
  "fr-FR": {
    "grid": {
      "EmptyRecord": "Aucun enregistrement à afficher",
      "True": "vrai",
      "False": "faux",
      "InvalidFilterMessage": "Données de filtre non valides",
      "GroupDropArea": "Faites glisser un en-tête de colonne ici pour
regrouper sa colonne",
      "UnGroup": "Cliquez ici pour dissocier",
      "GroupDisable": "Le regroupement est désactivé pour cette colonne",
      "FilterbarTitle": "Cellule de barre de filtre \"s\",
      "EmptyDataSourceError": "DataSource ne doit pas être vide lors du
chargement initial car les colonnes sont générées à partir de dataSource
dans AutoGenerate Column Grid",
      "Add": "Ajouter",
      "Edit": "Éditer",
      "Cancel": "Annuler",
      "Update": "Mise à jour",
      "Delete": "Supprimer",
      "Print": "Impression",
      "Pdfexport": "Exportation PDF",
      "Excelexport": "Exportation Excel",
      "Wordexport": "Exportation de mots",
      "Csvexport": "Exportation CSV",
      "Search": "Chercher",
      "Columnchooser": "Colonnes",
      "Save": "sauvegarder",
      "Item": "article",
      "Items": "articles",
      "EditOperationAlert": "Aucun enregistrement sélectionné pour
l'opération d'édition",
      "DeleteOperationAlert": "Aucun enregistrement sélectionné pour
l'opération de suppression",
      "SaveButton": "Sauvegarder",
      "OKButton": "Ok",
      "CancelButton": "Annuler",
      "EditFormTitle": "Les détails de",
      "AddFormTitle": "Ajouter un nouvel enregistrement",
      "BatchSaveConfirm": "Voulez-vous vraiment enregistrer les
modifications ?",
      "BatchSaveLostChanges": "Les modifications non enregistrées seront
perdues. Es-tu sur de vouloir continuer ?",
      "ConfirmDelete": "Voulez-vous vraiment supprimer l'enregistrement ?",
      "CancelEdit": "Voulez-vous vraiment annuler les modifications ?",
      "ChooseColumns": "Choisissez la colonne",
      "SearchColumns": "colonnes de recherche",
    }
  }
}

```

```

"Matches": "Aucun résultat",
"FilterButton": "Filtrer",
"ClearButton": "Effacer",
"StartsWith": "Commence par",
"EndsWith": "Se termine par",
"Contains": "Contient",
"Equal": "Égal",
"NotEqual": "Différent",
"LessThan": "Inférieur",
"LessThanOrEqual": "Inférieur ou égal",
"GreaterThan": "Supérieur",
"GreaterThanOrEqual": "Supérieur ou égal",
"ChooseDate": "Choisissez une date",
"EnterValue": "Entrez la valeur",
"Copy": "Copie",
"Group": "Regrouper par cette colonne",
"Ungroup": "Dissocier par cette colonne",
"autoFitAll": "Ajuster automatiquement toutes les colonnes",
"autoFit": "Ajuster automatiquement cette colonne",
"Export": "Exportation",
"FirstPage": "Première page",
"LastPage": "Dernière page",
"PreviousPage": "Page précédente",
"NextPage": "Page suivante",
"SortAscending": "Trier par ordre croissant",
"SortDescending": "Trier par ordre décroissant",
"EditRecord": "Modifier l'enregistrement",
>DeleteRecord": "Supprimer l'enregistrement",
"FilterMenu": "Filtre",
>SelectAll": "Tout sélectionner",
"Blanks": "Blancs",
"FilterTrue": "Vrai",
"FilterFalse": "Faux",
"NoResult": "Aucun résultat",
"ClearFilter": "Effacer le filtre",
"NumberFilter": "Filtres numériques",
"TextFilter": "Filtres de texte",
"DateFilter": "Filtres de date",
"DateTimeFilter": "Filtres DateTime",
"MatchCase": "Cas de correspondance",
"Between": "Entre",
"CustomFilter": "Filtre personnalisé",
"CustomFilterPlaceholder": "Entrez la valeur",
"CustomFilterDatePlaceholder": "Choisissez une date",
"AND": "ET",
"OR": "OU",
>ShowRowsWhere": "Afficher les lignes où:",
"NotStartsWith": "Ne commence pas par",
"Like": "Comme",
"NotEndsWith": "Ne se termine pas par",
"NotContains": "Ne contient pas",
"IsNull": "Nul",
"NotNull": "Non nul",
"IsEmpty": "Vide",
"IsNotEmpty": "Pas vide",
"AddCurrentSelection": "Ajouter la sélection actuelle au filtre",
"UnGroupButton": "Cliquez ici pour dissocier",

```

```

    "AutoFitAll": "Ajuster automatiquement toutes les colonnes",
    "AutoFit": "Ajuster automatiquement cette colonne",
    "Clear": "Dégager",
    "FilterMenuDialogARIA": "Boîte de dialogue du menu Filtre",
    "ExcelFilterDialogARIA": "Boîte de dialogue de filtre Excel",
    "DialogEditARIA": "Boîte de dialogue Modifier",
    "ColumnChooserDialogARIA": "Sélecteur de colonne",
    "ColumnMenuDialogARIA": "Boîte de dialogue du menu des colonnes",
    "CustomFilterDialogARIA": "Boîte de dialogue Filtre personnalisé",
    "SortAtoZ": "Trier de A à Z",
    "SortZtoA": "Trier de Z à A",
    "SortByOldest": "Trier par le plus ancien",
    "SortByNewest": "Trier par Plus récent",
    "SortSmallestToLargest": "Trier du plus petit au plus grand",
    "SortLargestToSmallest": "Trier du plus grand au plus petit",
    "Sort": "Sorte",
    "FilterDescription": "Appuyez sur Alt Bas pour ouvrir le menu du
filtre",
    "SortDescription": "Appuyez sur Entrée pour trier",
    "ColumnMenuDescription": "Appuyez sur Alt Bas pour ouvrir le menu des
colonnes",
    "GroupDescription": "Appuyez sur Ctrl espace pour grouper",
    "ColumnHeader": " en-tête de colonne ",
    "TemplateCell": " est une cellule modèle",
    "CommandColumnAria": "est l'en-tête de colonne de la colonne de
commande ",
    "DialogEdit": "Modifier la boîte de dialogue",
    "Clipboard": "presse-papiers",
    "GroupButton": "Bouton de groupe",
    "UnGroupAria": "bouton dissocier",
    "GroupSeperator": "Séparateur pour les colonnes groupées",
    "UnGroupIcon": "dissocier la colonne groupée ",
    "GroupedSortIcon": "trier la colonne groupée ",
    "GroupedDrag": "Faites glisser la colonne groupée",
    "GroupCaption": " est une cellule de légende de groupe",
    "CheckBoxLabel": "case à cocher",
    "Expanded": "Étendue",
    "Collapsed": "S'est effondré"
  },
  "pager": {
    "currentPageInfo": "{0} de {1} pages",
    "totalItemsInfo": "({0} éléments)",
    "firstPageTooltip": "Aller à la première page",
    "lastPageTooltip": "Aller à la dernière page",
    "nextPageTooltip": "Aller à la page suivante",
    "previousPageTooltip": "Aller à la page précédente",
    "nextPagerTooltip": "Aller aux éléments suivants du téléavertisseur",
    "previousPagerTooltip": "Accéder aux éléments de téléavertisseur
précédents",
    "pagerDropDown": "objets par page",
    "pagerAllDropDown": "Articles",
    "All": "Tout",
    "totalItemInfo": "({0} élément)",
    "Container": "Conteneur de téléavertisseur",
    "Information": "Informations sur le téléavertisseur",
    "ExternalMsg": "Message externe du téléavertisseur",
    "Page": "Page",
  }

```

```

        "Of": " de ",
        "Pages": " pages"
    }
}
}

```

### MAIN.TS

```

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

```

### Internationalization

The Internationalization library in the Syncfusion Angular Grid provides a localized display of number, date, and time values in the Grid component based on the preferred language and region.

[Internationalization](#) library allows you to globalize number, date, and time values using format strings defined in the [columns.format](#) property.

To work with **JSON** files in your application, you can enable JSON module resolution in TypeScript by adding the **resolveJsonModule** to true to your tsconfig.json file. Additionally, you can enhance module interoperability by setting **esModuleInterop** to true as shown below:

```

`ts
{
  compilerOptions: {
    resolveJsonModule: true,
    esModuleInterop: true,
  }
}
`

```

You need to load the culture format files corresponding to the desired locale in **ngOnInit** function. This ensures that the Grid component uses the correct format strings for number, date, and time values based on the selected culture. This can be demonstrated in the below example,

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService, PageService } from '@syncfusion/ej2-angular-grids'
import { L10n, loadCldr, setCulture, setCurrencyCode } from '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import cagregorian from './ca-gregorian.json';
import currencies from './currencies.json';
import numbers from './numbers.json';
import timeZoneNames from './timeZoneNames.json';

```



```

import numberingSystems from './numberingSystems.json'
import deDELocalization from './locale.json'
L10n.load(deDELocalization);
@Component({
  imports: [

    GridModule

  ],
  providers: [GroupService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[allowGrouping]="true" height='315px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='Freight' headerText='Freight'
[format]='formatOptions' textAlign='Right' width=80></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public formatOptions?: object;

  ngOnInit(): void {
    setCulture('de-DE');
    setCurrencyCode('EUR');
    loadCldr(
      cagregorian,
      currencies,
      numbers,
      timeZoneNames,
      numberingSystems
    );
    this.data = data;
    this.formatOptions = { format: 'C2', useGrouping: false,
minimumSignificantDigits: 1, maximumSignificantDigits: 3 }
  }
}

```

### LOCALE.JSON

```

{
  "de-DE": {
    "grid": {
      "EmptyRecord": "Keine Datensätze zum Anzeigen",
      "True": "wahr",
      "False": "falsch",
      "InvalidFilterMessage": "Ungültige Filterdaten",
      "GroupDropArea": "Ziehen Sie eine Spaltenüberschrift hierher, um die
Spalte zu gruppieren",

```

```

    "UnGroup": "Klicken Sie hier, um die Gruppierung aufzuheben",
    "GroupDisable": "Die Gruppierung ist für diese Spalte deaktiviert",
    "FilterbarTitle": "\"s Filterbalkenzelle",
    "EmptyDataSourceError": "Datenquelle darf beim ersten Laden nicht
    leer sein, da Spalten aus der Datenquelle automatisch generiert werden",
    "Add": "Hinzufügen",
    "Edit": "Bearbeiten",
    "Cancel": "Abbrechen",
    "Update": "Aktualisieren",
    "Delete": "Löschen",
    "Print": "Drucken",
    "Pdfexport": "PDF-Export",
    "Excelexport": "Excel-Export",
    "Wordexport": "Word-Export",
    "Csvexport": "CSV-Export",
    "Search": "Suchen",
    "Columnchooser": "Spalten",
    "Save": "Speichern",
    "Item": "Datensatz",
    "Items": "Datensätze",
    "EditOperationAlert": "Keine Datensätze zum Bearbeiten ausgewählt",
    "DeleteOperationAlert": "Keine Datensätze zum Löschen ausgewählt",
    "SaveButton": "Speichern",
    "OKButton": "in Ordnung",
    "CancelButton": "Abbrechen",
    "EditFormTitle": "Details von",
    "AddFormTitle": "Neuen Datensatz hinzufügen",
    "BatchSaveConfirm": "Möchten Sie die Änderungen wirklich
    speichern?",
    "BatchSaveLostChanges": "Nicht gespeicherte Änderungen gehen
    verloren. Sind Sie sicher, dass Sie fortfahren wollen?",
    "ConfirmDelete": "Möchten Sie den Datensatz wirklich löschen?",
    "CancelEdit": "Möchten Sie die Änderungen wirklich abbrechen?",
    "ChooseColumns": "Wählen Sie Spalten",
    "SearchColumns": "Spalten durchsuchen",
    "Matches": "Keine Treffer gefunden",
    "FilterButton": "Filter",
    "ClearButton": "Löschen",
    "StartsWith": "Beginnt mit",
    "EndsWith": "Endet mit",
    "Contains": "Enthält",
    "Equal": "Gleich",
    "NotEqual": "Nicht gleich",
    "LessThan": "Weniger als",
    "LessThanOrEqual": "Weniger als oder gleich",
    "GreaterThan": "Grösser als",
    "GreaterThanOrEqual": "Grösser als oder gleich",
    "ChooseDate": "Wählen Sie ein Datum",
    "EnterValue": "Geben Sie den Wert ein",
    "Copy": "Kopieren",
    "Group": "Nach dieser Spalte gruppieren",
    "Ungroup": "Gruppierung nach dieser Spalte aufheben",
    "autoFitAll": "Automatisch alle Spalten anpassen",
    "autoFit": "Diese Spalte automatisch anpassen",
    "Export": "Export",
    "FirstPage": "Erste Seite",
    "LastPage": "Letzte Seite",

```

```

"PreviousPage": "Vorherige Seite",
"NextPage": "Nächste Seite",
"SortAscending": "Aufsteigend sortieren",
"SortDescending": "Absteigend sortieren",
"EditRecord": "Datensatz bearbeiten",
"DeleteRecord": "Datensatz löschen",
"FilterMenu": "Filter",
"SelectAll": "Wählen Sie Alle",
"Blanks": "Leerzeichen",
"FilterTrue": "Wahr",
"FilterFalse": "Falsch",
"NoResult": "Keine Treffer gefunden",
"ClearFilter": "Filter löschen",
"NumberFilter": "Anzahl Filter",
"TextFilter": "Textfilter",
"DateFilter": "Datumsfilter",
"DateTimeFilter": "DatumsZeit-Filter",
"MatchCase": "Gross- / Kleinschreibung",
"Between": "Zwischen",
"CustomFilter": "Benutzerdefinierte Filter",
"CustomFilterPlaceholder": "Geben Sie einen Wert ein",
"CustomFilterDatePlaceholder": "Wählen Sie ein Datum",
"AND": "UND",
"OR": "ODER",
"ShowRowsWhere": "Zeilen anzeigen, in denen:",
"NotStartsWith": "beginnt nicht mit",
"Like": "Wie",
"NotEndsWith": "endet nicht mit",
"NotContains": "enthält nicht",
"IsNull": "Null",
"NotNull": "nicht null",
"IsEmpty": "leer",
"IsNotEmpty": "nicht leer",
"AddCurrentSelection": "Aktuelle Auswahl zum Filter hinzufügen",
"UnGroupButton": "Klicken Sie hier, um die Gruppierung aufzuheben",
"AutoFitAll": "Automatisch alle Spalten anpassen",
"AutoFit": "Diese Spalte automatisch anpassen",
"Clear": "Klar",
"FilterMenuDialogARIA": "Menüdialog filtern",
"ExcelFilterDialogARIA": "Excel-Filterdialog",
"DialogEditARIA": "Dialog bearbeiten",
"ColumnChooserDialogARIA": "Spaltenauswahl",
"ColumnMenuDialogARIA": "Spaltenmenüdialog",
"CustomFilterDialogARIA": "Benutzerdefinierter Filterdialog",
"SortAtoZ": "Sortiere von A bis Z",
"SortZtoA": "Z bis A sortieren",
"SortByOldest": "Nach Ältesten sortieren",
"SortByNewest": "Nach Neuesten sortieren",
"SortSmallestToLargest": "Vom Kleinsten zum Größten sortieren",
"SortLargestToSmallest": "Vom Größten zum Kleinsten sortieren",
"Sort": "Sortieren",
"FilterDescription": "Drücken Sie die Alt-Nach-unten-Taste, um das Filtermenü zu öffnen",
"SortDescription": "Drücken Sie zum Sortieren die Eingabetaste",
"ColumnMenuDescription": "Drücken Sie die Alt-Nach-unten-Taste, um das Spaltenmenü zu öffnen",
"GroupDescription": "Drücken Sie zum Gruppieren die Strg-Leertaste",

```

```

    "ColumnHeader": " Spaltenüberschrift ",
    "TemplateCell": " ist eine Vorlagenzelle",
    "CommandColumnAria": "ist die Spaltenüberschrift der Befehlsspalte
",
    "DialogEdit": "Dialog bearbeiten",
    "ClipBoard": "Zwischenablage",
    "GroupButton": "Gruppenschaltfläche",
    "UnGroupAria": "Schaltfläche zum Aufheben der Gruppierung",
    "GroupSeperator": "Trennzeichen für die gruppierten Spalten",
    "UnGroupIcon": "heben Sie die Gruppierung der gruppierten Spalte auf
",
    "GroupedSortIcon": "sortieren Sie die gruppierte Spalte ",
    "GroupedDrag": "Ziehen Sie die gruppierte Spalte",
    "GroupCaption": " ist eine Gruppenunterschriftszelle",
    "CheckBoxLabel": "Kontrollkästchen",
    "Expanded": "Erweitert",
    "Collapsed": "Zusammengebrochen"
},
"pager": {
    "currentPageInfo": "{0} von {1} Seiten",
    "totalItemsInfo": "({0} Datensätze)",
    "firstPageTooltip": "Gehe zur ersten Seite",
    "lastPageTooltip": "Gehe zur letzten Seite",
    "nextPageTooltip": "Gehe zur nächsten Seite",
    "previousPageTooltip": "Zurück zur letzten Seite",
    "nextPagerTooltip": "Gehe zu den nächsten Pager-Elementen",
    "previousPagerTooltip": "Gehen Sie zu den vorherigen Pager-
Elementen",
    "pagerDropDown": "Datensätze pro Seite",
    "pagerAllDropDown": "Datensätze",
    "All": "Alle",
    "totalItemInfo": "({0} Datensatz)",
    "Container": "Pager-Container",
    "Information": "Pager-Informationen",
    "ExternalMsg": "Pager-externe Nachricht",
    "Page": "Buchseite ",
    "Of": " von ",
    "Pages": " Seiten"
}
}
}

```

### 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, **Freight** column is formatted using the [NumberFormatOptions](#).

\* By default, [locale](#) value is **en-US**. If you wish to change the culture to something other than **en-US**, you can simply set the `locale` property accordingly.

### Right to Left - RTL

The Right to Left (RTL) feature in the Syncfusion Angular Grid allows you to switch the text direction and layout from left-to-right to right-to-left. This feature is especially beneficial for interacting with the grid in languages that are written and read from right to left, such as **Arabic, Farsi, Urdu**, and others. Enabling RTL significantly improves the experience and accessibility for such languages.

To enable RTL in the Grid, you need to set the [enableRtl](#) property to **true**. By setting `enableRtl`, the grid component's text direction and layout will be adjusted to support right-to-left languages.

To work with **JSON** files in your application, you can enable JSON module resolution in TypeScript by adding the `resolveJsonModule` to true to your `tsconfig.json` file. Additionally, you can enhance module interoperation by setting `esModuleInterop` to true as shown below:

```
`ts
{
  compilerOptions: {
    resolveJsonModule: true,
    esModuleInterop: true,
  }
}
```

In the following example, the [EJ2 Toggle Switch Button](#) component is added to enable or disable the Right to Left (RTL) feature for the **Arabic (ar-AE)** locale. When the switch is toggled, the [change](#) event is triggered and the `enableRtl` property of the grid is updated accordingly.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import {
  ButtonModule,
  CheckBoxModule,
  RadioButtonModule,
  SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { GridModule, PageService, ToolbarService, EditService, GroupService,
  FilterService, SortService, ReorderService, ColumnMenuService,
  ColumnChooserService } from '@syncfusion/ej2-angular-grids'
import { L10n, setCulture } from '@syncfusion/ej2-base';
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from '../datasource';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
import { GridComponent, PageSettingsModel, ToolbarItems, EditSettingsModel }
  from '@syncfusion/ej2-angular-grids';
import arAELocalization from '../locale.json';
L10n.load(arAELocalization);
@Component({
  imports: [
    GridModule,
```

```

        ButtonModule,
        CheckBoxModule,
        RadioButtonModule,
        SwitchModule,
    ],
    providers: [PageService, ToolbarService, EditService,
        GroupService, FilterService, SortService,
        ReorderService, ColumnMenuService, ColumnChooserService],
    standalone: true,
    selector: 'app-root',
    template: `
        <div>
            <label style="padding: 10px 10px">
                Enable or diable RTL mode
            </label>
            <ejs-switch id="switch" (change)="onSwitchChange($event)"></ejs-switch>
        </div>
        <ejs-grid #grid [dataSource]='data' [allowSorting]='true'
[allowReordering]='true'
    [allowFiltering]='true' [allowGrouping]='true'
[editSettings]="editSettings" [toolbar]="toolbar"
    [enableRtl]='true' [locale]='locale' [allowPaging]='true'
[pageSettings]='pageOptions'
    [showColumnMenu]='true' [showColumnChooser]='true'>
            <e-columns>
                <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=90></e-column>
                <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
                <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
                <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
            </e-columns>
        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public pageOptions?: PageSettingsModel;
        public locale: any = 'ar-AE';
        public editSettings?: EditSettingsModel;
        public toolbar?: ToolbarItems[];
        @ViewChild('grid')
        public grid?: GridComponent;
        ngOnInit(): void {
            this.data = data;
            this.pageOptions = { pageSize: 7 };
            this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Normal' };
            this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        }
        onSwitchChange(args: ChangeEventArgs) {
            if (args.checked) {
                (this.grid as GridComponent).enableRtl = false;
            } else {
                (this.grid as GridComponent).enableRtl = true;
            }
        }
    }

```

```
}
}
```

## LOCALE.JSON

```
{
  "ar-AE": {
    "grid": {
      "EmptyRecord": "لا سجلات لعرضها",
      "True": "صحيح",
      "False": "خاطئة",
      "InvalidFilterMessage": "بيانات تصفية غير صالحة",
      "GroupDropArea": "اسحب رأس العمود هنا لتجميع العمود",
      "UnGroup": "انقر هنا لإلغاء التجميع",
      "GroupDisable": "تم تعطيل التجميع لهذا العمود",
      "FilterbarTitle": "خلية شريط مرشح",
      "EmptyDataSourceError": "فارغًا عند التحميل DataSource يجب ألا يكون",
      "Add": "إضافة",
      "Edit": "تعديل",
      "Cancel": "إلغاء",
      "Update": "تحديث",
      "Delete": "حذف",
      "Print": "طباعة",
      "Pdfexport": "تصدير قوات الدفاع الشعبي",
      "Excelexport": "اكسل التصدير",
      "Wordexport": "كلمة تصدير",
      "Csvexport": "تصدير CSV",
      "Search": "بحث",
      "Columnchooser": "أعمدة",
      "Save": "حفظ",
      "Item": "بند",
      "Items": "العناصر",
      "EditOperationAlert": "لم يتم تحديد سجلات لعملية التحرير",
      "DeleteOperationAlert": "لم يتم تحديد سجلات لعملية الحذف",
      "SaveButton": "حفظ",
      "OKButton": "حسنًا",
      "CancelButton": "إلغاء",
      "EditFormTitle": "تفاصيل",
      "AddFormTitle": "إضافة سجل جديد",
      "BatchSaveConfirm": "هل أنت متأكد أنك تريد حفظ التغييرات؟",
      "BatchSaveLostChanges": "سيتم فقد التغييرات غير المحفوظة. هل أنت متأكد أنك تريد المتابعة؟",
      "ConfirmDelete": "هل أنت متأكد أنك تريد حذف السجل؟",
      "CancelEdit": "هل أنت متأكد من أنك تريد إلغاء التغييرات؟",
      "ChooseColumns": "اختيار العمود",
      "SearchColumns": "أعمدة البحث",
      "Matches": "لم يتم العثور على تطابق",
      "FilterButton": "منقي",
      "ClearButton": "واضح",
      "StartsWith": "ابدا بـ",
      "EndsWith": "ينتهي بـ",
      "Contains": "يحتوي على",
      "Equal": "مساو",
      "NotEqual": "ليس متساوي",
      "LessThan": "أقل من",
    }
  }
}
```

```

"LessThanOrEqual": "أصغر من أو يساوي",
"GreaterThan": "أكثر من",
"GreaterThanOrEqual": "أكبر من أو يساوي",
"ChooseDate": "اختيار التاريخ",
"EnterValue": "أدخل القيمة",
"Copy": "نسخ",
"Group": "تجميع حسب هذا العمود",
"Ungroup": "فك تجميع بواسطة هذا العمود",
"autoFitAll": "احتواء تلقائي لجميع الأعمدة",
"autoFit": "احتواء تلقائي لهذا العمود",
"Export": "تصدير",
"FirstPage": "الصفحة الأولى",
"LastPage": "آخر صفحة",
"PreviousPage": "الصفحة السابقة",
"NextPage": "الصفحة التالية",
"SortAscending": "فرز تصاعدي",
"SortDescending": "ترتيب تنازلي",
"EditRecord": "تحرير السجل",
"DeleteRecord": "حذف سجل",
"FilterMenu": "منقي",
"SelectAll": "اختر الكل",
"Blanks": "الفراغات",
"FilterTrue": "صحيح",
"FilterFalse": "خاطئة",
"NoResult": "لم يتم العثور على تطابق",
"ClearFilter": "مرشح واضح",
"NumberFilter": "عدد المرشحات",
"TextFilter": "مرشحات النص",
"DateFilter": "مرشحات التاريخ",
"DateTimeFilter": "Date Time مرشحات",
"MatchCase": "حالة مباراة",
"Between": "ما بين",
"CustomFilter": "تصفية مخصص",
"CustomFilterPlaceholder": "أدخل القيمة",
"CustomFilterDatePlaceholder": "اختيار موعد",
"AND": "و",
"OR": "أو",
"ShowRowsWhere": "إظهار الصفوف حيث",
"NotStartsWith": "لا تبدأ بـ",
"Like": "يحب",
"NotEndsWith": "لا تنتهي بـ",
"NotContains": "لا يحتوي",
"IsNull": "باطل",
"NotNull": "غير فارغة",
"IsEmpty": "فارغ",
"IsNotEmpty": "ليس فارغاً",
"AddCurrentSelection": "إضافة التحديد الحالي للتصفية",
"UnGroupButton": "انقر هنا لفك التجميع",
"AutoFitAll": "احتواء تلقائي لجميع الأعمدة",
"AutoFit": "احتواء تلقائي لهذا العمود",
"Clear": "صافي",
"FilterMenuDialogARIA": "مربع حوار قائمة التصفية",
"ExcelFilterDialogARIA": "Excel مربع حوار مرشح",
"DialogEditARIA": "مربع حوار التحرير",
"ColumnChooserDialogARIA": "منتقي الأعمدة",
"ColumnMenuDialogARIA": "مربع حوار قائمة العمود",
"CustomFilterDialogARIA": "مربع حوار عامل التصفية المخصص",

```



```

"SortAtoZ": "فرز من الألف إلى الياء",
"SortZtoA": "A إلى Z فرز من",
"SortByOldest": "فرز حسب الأقدم",
"SortByNewest": "فرز حسب الأحدث",
"SortSmallestToLargest": "الفرز من الأصغر إلى الأكبر",
"SortLargestToSmallest": "الفرز من الأكبر إلى الأصغر",
"Sort": "فرز",
"FilterDescription": "الفتح قائمة التصفية Alt Down اضغط على",
"SortDescription": "للفرز Enter اضغط على",
"ColumnMenuDescription": "لأسفل لفتح قائمة العمود Alt اضغط على",
"GroupDescription": "مسافة للمجموعة Ctrl اضغط على مفتاح",
"ColumnHeader": "رأس العمود",
"TemplateCell": "هي خلية قالب",
"CommandColumnAria": "هو رأس عمود الأمر",
"DialogEdit": "تحرير الحوار",
"ClipBoard": "الحافظة",
"GroupButton": "زر المجموعة",
"UnGroupAria": "زر فك التجميع",
"GroupSeperator": "فاصل للأعمدة المجمعة",
"UnGroupIcon": "قم بفك تجميع العمود المجمع",
"GroupedSortIcon": "قم بفرز العمود المجمع",
"GroupedDrag": "اسحب العمود المجمع",
"GroupCaption": "هي خلية تسمية توضيحية جماعية",
"CheckBoxLabel": "خانة الاختيار",
"Expanded": "موسع",
"Collapsed": "انهار"
},
"pager": {
  "currentPageInfo": "من {0} 1 {صفحة}",
  "totalItemsInfo": "(عناصر {0})",
  "firstPageTooltip": "الذهاب إلى الصفحة الأولى",
  "lastPageTooltip": "الذهاب إلى الصفحة الأخيرة",
  "nextPageTooltip": "انتقل إلى الصفحة التالية",
  "previousPageTooltip": "الانتقال إلى الصفحة السابقة",
  "nextPagerTooltip": "انتقل إلى عناصر بيكر التالية",
  "previousPagerTooltip": "انتقل إلى عناصر بيكر السابقة",
  "pagerDropDown": "مواد لكل صفحة",
  "pagerAllDropDown": "العناصر",
  "All": "الكل",
  "totalItemInfo": "(بند {0})",
  "Container": "حاوية بيكر",
  "Information": "معلومات جهاز النداء",
  "ExternalMsg": "رسالة خارجية بيكر",
  "Page": "صفحة",
  "Of": "من",
  "Pages": "الصفحات"
}
}
}

```

## 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](#)
- [Animate the Grid selected row in Angular Grid](#)

## Accessibility in Angular Grid component

The Grid component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.


The accessibility compliance for the Grid component is outlined below.


| Accessibility Criteria | Compatibility |


| -- | -- |


| [WCAG 2.2 Support](#) |  |  
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Section 508 Support](#) |  |  
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Screen Reader Support](#) |  |  
page/yes.png" alt="Yes"> |

| [Right-To-Left Support](#) |  |  
alt="Yes"> |

| [Color Contrast](#) |  |  
alt="Yes"> |

| [Mobile Device Support](#) |  |  
page/yes.png" alt="Yes"> |

| [Keyboard Navigation Support](#) |  |  
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Accessibility Checker Validation](#) |  |  
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

| [Axe-core Accessibility Validation](#) |  |  
src="https://cdn.syncfusion.com/content/images/documentation/partial.png" alt="Intermediate"> |

<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 Grid component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Grid component:

| Attributes | Purpose |

| --- | --- |

| **role=grid** | To represent the element containing the grid component. |

| **role=row** | To represent the element containing the cells of the row in the grid. |

| **role=rowgroup** | To represent the group of rows in the grid. |

| **role=columnheader** | To represent the cell in a row contains header information for a column in the grid. |

| **role=gridcell** | To represent a cell in the grid component. |

| **role=button** | To represent the element that acts as a button in the grid. |

| **role=search** | To represent the element that acts as a search region in the grid. |

| **role=presentation** | To represent the element to be not available for accessibility concerns. |

| **role=navigation** | To represent the element containing pager elements to navigate from one page to another. |

| **aria-colindex** | Defines the column index of the column with respect to the total number of columns within the grid. |

| **aria-rowindex** | Defines row index of the row with respect to the total number of rows within the grid. |

| **aria-rowspan** | Defines the number of rows spanned by a cell within the grid. |

| **aria-colspan** | Defines the number of columns spanned by a cell within the grid. |

| **aria-rowcount** | Defines the total number of rows in the grid. |

| **aria-colcount** | Defines the total number of columns in the grid. |

| **aria-selected** | Indicates the current "selected" state of the rows and cells in the grid. |

| **aria-expanded** | Indicate if the expand icon in the hierarchy grid or grouped grid or detail grid is expanded or collapsed |

| **aria-sort** | Indicates whether the data in the grid are sorted in ascending or descending order. |

| **aria-busy** | Indicates an element is being modified and that assistive technologies may want to wait until the changes are complete before informing the user about the update. |

| **aria-owns** | Identifies an element in order to define a visual, functional, or contextual relationship between a parent and its child elements. |

| **aria-hidden** | Hides the element from accessibility concerns. |

| **aria-labelledby** | Provides an accessible name for the checkbox labels in excel filter, checkbox filter and column chooser dialog. |

| **aria-describedby** | Provides an description about the features enabled in the header when the grid header cell is focused. |

The Syncfusion Grid component is structured with a two-table architecture for its header and content. To enhance accessibility for screen readers, roles and ARIA attributes are incorporated for both the grid parent and all its child elements. Although this architectural approach may have some limitations with accessibility checker tools. It's important to note that these limitations do not affect the readability of the grid content over screen readers.

The accessibility checker tools highlights the following known issues:

- **aria-required-children**: This warning appears when rendering the grid without any features, as it contains textarea and grid content. Additionally, it appears when enabling features such as the toolbar and grouping.
- **color-contrast**: This warning appears when you are enabling the search item in the grid's toolbar.
- An explicit ARIA 'role' is not valid for element within an ARIA role 'grid' per the ARIA in HTML specification.
- An explicit ARIA 'role' is not valid for element within an ARIA role 'grid' per the ARIA in HTML specification.
- An explicit ARIA 'role' is not valid for element within an ARIA role 'grid' per the ARIA in HTML specification.
- The element with role "button" contains descendants with roles "rowgroup" which are ignored by browsers.
- Content is not within a landmark element.
- Multiple elements with "search" role do not have unique labels.
- Text contrast of 4.10 with its background is less than the WCAG AA minimum requirements for text of size 13px and weight of 400.
- Interactive component with ARIA role 'grid' does not have a programmatically associated name.
- The element with role "rowgroup" is not contained in or owned by an element with one of the following roles: "grid, table, treegrid".

### Keyboard interaction

The Grid component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Grid component.

<b>Pager</b>

**Windows | MAC | To do this**

**Home | Fn + Left Arrow | Moves the focus to the first cell of the focused row.**

**End | Fn + Right Arrow | Moves the focus to the last cell of the focused row.**

**Ctrl + Home | Command + Fn + Left Arrow | Moves the focus to the first Cell of the first row in the grid.**

**Ctrl + End | Command + Fn + Right Arrow | Moves the focus to the last Cell of the last row in the grid.**

**Up Arrow | Up Arrow | Moves the cell focus upward from the focused cell.**

**Down Arrow | Down Arrow | Moves the cell focus downward from the focused cell.**

**Right Arrow | Right Arrow | Moves the cell focus right side from the focused cell.**

**Left Arrow | Left Arrow | Moves the cell focus left side from the focused cell.**

**Alt + J | Alt + J | Moves the focus to the entire grid.**

**Alt + W | Alt + W | Move the focus to the grid content element.**

<b>Selection</b>

**Windows | MAC | To do this**

**Ctrl + Up Arrow | Command + Up Arrow | Collapses all the visible groups.**

**Ctrl + Down Arrow | Command + Down Arrow | Expands all the visible groups.**

**Ctrl + Space | Ctrl + Space | Performs grouping when focused on a header element.**

**Enter | Enter | If the current cell is an expand/collapse cell then expands/collapses the current group/detailrow/childgrid.**

<b>Print</b>

**Windows | MAC | To do this**

**Ctrl + C | Command + C | Copies selected rows or cells data into the clipboard.**

**Ctrl + Shift + H | Ctrl + Shift + H | Copies selected rows or cells data with header into clipboard**

<b>Editing</b>

**Windows | MAC | To do this**

**Alt + Down arrow | Alt + Down arrow | Opens the filter menu(excel, menu and checkbox filter) when its header element is in focused state.**

<b>Column Menu</b>

**Windows | MAC | To do this**

**Ctrl + left arrow or right arrow | Command + left arrow or right arrow | Reorders the focused header column to the left or right side.**

<b>Sorting</b>

**Windows | MAC | To do this**

Enter | Enter | Performs sorting(ascending/descending) on a column when its header element is in focused state.

Ctrl + Enter | Command + Enter | Performs multi-sorting on a column when its header element is in focused state.

Shift + Enter | Shift + Enter | Clears sorting for the focused header column.

<br>

\* The Command and Control keys on Mac devices can be interchanged. When this switch occurs, use the Command key in place of the Control key and the Control key in place of the Command key for the above listed key interactions with Mac devices. For example, after switching the keys to group the columns when the header element is focused use Command + Space and for expanding the visible groups use Ctrl + Down Arrow.

#### *How to prevent default key action behavior*

The Syncfusion Angular Grid provides flexibility to prevent the default key action behavior based on your requirements. This enables you to intercept and customize the behavior when specific keys are pressed within a web application

To prevent the default key action behavior in the grid, you can utilize the [keyPressed](#) event. This event is triggered for every key press, allowing you to customize the behavior based on the pressed key.

The following example demonstrates how to prevent the default behavior of the “ENTER” key using the `keyPressed` event.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, SortService, FilterService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { KeyboardEventArgs } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService, SortService, FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data'
(keyPressed)="keyPressed($event)">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>`
})
export class AppComponent {
  ngOnInit() {
  }
}
```

```

        </ejs-grid>`
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        ngOnInit(): void {
            this.data = data;
        }
        keyPressed(args : KeyboardEventArgs) {
            if (args.keyCode === 13) {
                // Prevent the default Enter key action
                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));

```

#### Custom shortcut keys to perform grid actions

The Syncfusion Angular Grid component enables you to enhance the usability of keyboard shortcuts for various grid actions and navigation. In addition to the built-in keyboard navigation capabilities, you can implement custom keyboard shortcuts to execute specific actions.

To achieve this, you can utilize the [keyPressed](#) event of the grid. This event is triggered for every key press, allowing you to customize the behavior based on the pressed key.

The following example demonstrates how to perform grid actions using shortcut keys through the [keyPressed](#) event. Within the event, define the following custom shortcuts to perform various grid actions:

- Pressing N adds a new record.
- Pressing Ctrl + S save a record by invoking endEdit.
- Pressing Ctrl + D deletes a record.
- Pressing Ctrl + A selects all rows.
- Pressing Ctrl + G groups the grid by a specified column.

And prevented the default actions associated with the following keyboard shortcuts used for default grouping and editing action:

- Ctrl + Space
- Insert
- F2
- Delete
- Enter

You can add more custom shortcuts and actions as needed to enhance the functionality of your Syncfusion Angular Grid component.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { cascadeData } from './datasource';
import { EditSettingsModel, ToolbarItems, GridComponent,
SelectionSettingsModel, KeyboardEventArgs } from '@syncfusion/ej2-angular-
grids';
@Component({
imports: [

        GridModule,
        FormsModule,

    ],
providers: [EditService, ToolbarService, GroupService],
standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #grid [dataSource]='data'
(keyPressed)="keyPressed($event)" [editSettings]='editSettings'
[toolbar]='toolbar' height='273px' allowGrouping="true"
[selectionSettings]='selectionOptions' >
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
            <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
        </e-columns>
    </ejs-grid>`
})
export class AppComponent implements OnInit {
    public data?: object[];
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItems[];
    public selectionOptions?: SelectionSettingsModel;
    @ViewChild('grid') grid?: GridComponent;
    keyPressed(e: KeyboardEventArgs) {
        const key = e.key.toLowerCase();
        switch (key) {
            case 'n':
                e.preventDefault();
                (this.grid as GridComponent).addRecord();
                break;
            case 's':
                if (e.ctrlKey) {
                    e.preventDefault();
                    (this.grid as GridComponent).endEdit();
                }
                break;
            case 'd':
                if (e.ctrlKey) {
                    e.preventDefault();

```



```

        (this.grid as GridComponent).deleteRecord();
    }
    break;
case 'a':
    if (e.ctrlKey) {
        e.preventDefault();
        (this.grid as GridComponent).selectRowsByRange(0);
    }
    break;
case 'g':
    if (e.ctrlKey) {
        e.preventDefault();
        (this.grid as GridComponent).groupColumn('CustomerID');
    }
    break;
case 'enter':
    e.preventDefault();
    e.cancel = true;
    (this.grid as GridComponent).refreshColumns();
    break;
case 'insert':
    e.preventDefault();
    e.cancel = true;
    break;
case 'delete':
    e.preventDefault();
    e.cancel = true;
    break;
case 'f2':
    e.preventDefault();
    e.cancel = true;
    break;
case '" "':
    if (e.ctrlKey) {
        e.preventDefault();
        e.cancel = true;
    }
    break;
    // Add more custom shortcuts as needed
}
}
ngOnInit(): void {
    this.data = cascadeData;
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    this.selectionOptions = {
        type: 'Multiple',
    };
}
}
}

```

**MAIN.TS**

```

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

```

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

### Ensuring accessibility

The Grid component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Grid component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Grid component with accessibility tools.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, ReorderService, FilterService,
GroupService, ColumnChooserService,
AggregateService, ToolbarService, SelectionService, RowDDService,  } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { PageSettingsModel, ToolbarItems, SearchSettingsModel,
SelectionSettingsModel,
    FilterSettingsModel, GroupSettingsModel, EditSettingsModel,
SortSettingsModel } from '@syncfusion/ej2-angular-grids';

@Component({
imports: [

    GridModule

],
providers: [PageService,
    SortService,
    ReorderService,
    FilterService,
    GroupService,
    AggregateService,
    ToolbarService,
    SelectionService,
    RowDDService,
    ColumnChooserService ],
standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' [allowPaging]="true"
[pageSettings]="pageSettings" [searchSettings]='searchOptions'
[toolbar]='toolbarOptions' [allowReordering]='true' [allowSorting]="true"
[allowReordering]='true' [allowRowDragAndDrop]='true'
[selectionSettings]='selectionOptions' [selectedRowIndex]='6'
[allowSorting]="true" [sortSettings]='sortOptions'
[allowFiltering]="true" [filterSettings]='filterOptions'
[allowGrouping]="true" [groupSettings]='groupOptions'
[editSettings]='editSettings' [showColumnChooser]= 'true'>
    <e-columns>
        <e-column type="checkbox" width=50></e-column>
```

```

        <e-column field='OrderID' headerText='Order ID'
isPrimaryKey='true' textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
        <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
        <e-column field='Freight' headerText='Freight($)'
textAlign='Right' format='C2' width=90></e-column>
        <e-column field='ShipCity' headerText='Ship City' width=150></e-
column>
        <e-column field='ShipCountry' headerText='Ship Country'
width=140></e-column>
        <e-column field='ShipName' headerText='Shipped Name'
textAlign='Right' width=140></e-column>
    </e-columns>
    <e-aggregates>
        <e-aggregate>
            <e-columns>
                <e-column field="Freight" type="sum">
                    <ng-template #footerTemplate let-data>Sum:
{{data.sum}} </ng-template>
                </e-column>
            </e-columns>
        </e-aggregate>
        <e-aggregate>
            <e-columns>
                <e-column field="Freight" type="sum">
                    <ng-template #groupFooterTemplate let-data>Sum:
{{data.sum}} </ng-template>
                </e-column>
            </e-columns>
        </e-aggregate>
        <e-aggregate>
            <e-columns>
                <e-column field="Freight" type="max">
                    <ng-template #groupCaptionTemplate let-data>Max:
{{data.max}}</ng-template>
                </e-column>
            </e-columns>
        </e-aggregate>
    </e-aggregates>
</ejs-grid>`
    ))
export class AppComponent implements OnInit {
    public data?: object[];
    public pageSettings?: PageSettingsModel;
    public toolbarOptions?: ToolbarItems[];
    public searchOptions?: SearchSettingsModel;
    public selectionOptions?: SelectionSettingsModel;
    public sortOptions?: SortSettingsModel;
    public filterOptions?: FilterSettingsModel;
    public groupOptions?: GroupSettingsModel;
    public editSettings?: EditSettingsModel;
    ngOnInit(): void {
        this.data = data;
        this.pageSettings = { pageCount: 2, pageSizes: true };
    }
}

```

```

        this.searchOptions = { fields: ['ShipCountry'], operator:
'contains', key: 'a', ignoreCase: true };
        this.toolbarOptions = ['Add', 'Edit', 'Delete', 'Update', 'Cancel',
'Search', 'ColumnChooser'];
        this.selectionOptions = { type: 'Multiple', mode: 'Both' };
        this.sortOptions = { columns: [{ field: 'OrderID', direction:
'Ascending' }, { field: 'ShipCity', direction: 'Descending' }] };
        this.filterOptions = {type:'Excel'};
        this.groupOptions = { columns: ['CustomerID'] };
        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: 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

- [Accessibility in Syncfusion Angular components](#)

### Clipboard in Angular Grid component

The clipboard feature in the Syncfusion Angular Grid provides an easy way to copy selected rows or cells data into the clipboard. You can use keyboard shortcuts to perform the copy operation. The following list of keyboard shortcuts is supported in the Grid to copy selected rows or cells data into clipboard.

Interaction keys | Description

**Ctrl + C | Copy selected rows or cells data into clipboard.**

**Ctrl + Shift + H | Copy selected rows or cells data with header into clipboard.**

By using these keyboard shortcuts, you can quickly copy data from the grid to the clipboard, making it easy to paste the data into other applications or documents.

To enable the clipboard feature, you can use the grid component with your data source and selection property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],

```

```

standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='272px'
[selectionSettings]='selectionOptions'>
    <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
        <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
        <e-column field='ShipName' headerText='Ship Name'
width=130></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public selectionOptions?: SelectionSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.selectionOptions = { type: 'Multiple' };
  }
}

```

### MAIN.TS

```

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

```

### Copy to clipboard by external buttons

Copying data to the clipboard by using external buttons in the Syncfusion Angular Grid allows you to programmatically trigger the copy operation, making it more friendly, especially for those who may not be familiar with keyboard shortcuts or manual copying.

To copy selected rows or cells data into the clipboard with the help of external buttons, you can utilize the [copy](#) method available in the grid component. This is demonstrated in the following example,

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { SelectionSettingsModel, GridComponent } from '@syncfusion/ej2-
angular-grids';
@Component({
  imports: [

    GridModule,
    ButtonModule

```

```

    ],
    standalone: true,
    selector: 'app-root',
    template: `
      <button ejs-button id='copy' cssClass="e-outline"
      (click)='copy()'>Copy</button>
      <button ejs-button id='copyHeader' cssClass="e-outline"
      (click)='copyHeader()' style="margin-left:10px">CopyHeader</button>
      <ejs-grid #grid style="padding: 5px 5px" id='grid' [dataSource]='data'
      height='280px' [selectionSettings]='selectionOptions'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
          textAlign='Right' width=90></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
          width=100></e-column>
          <e-column field='ShipCity' headerText='Ship City' width=100></e-
          column>
          <e-column field='ShipName' headerText='Ship Name' width=120></e-
          column>
        </e-columns>
      </ejs-grid>`
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    public selectionOptions?: SelectionSettingsModel;
    @ViewChild('grid') public grid?: GridComponent;
    ngOnInit(): void {
      this.data = data;
      this.selectionOptions = { type: 'Multiple' };
    }
    copy() {
      (this.grid as GridComponent).copy();
    }
    copyHeader() {
      (this.grid as GridComponent).copy(true);
    }
  }
}

```

## MAIN.TS

```

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

```

## AutoFill

The AutoFill feature in the Syncfusion Angular Grid allows you to copy the data of selected cells and paste it into other cells by simply dragging the autofill icon of the selected cells to the desired cells. This feature provides a convenient way to quickly populate data in a grid.

### how to use the autofill feature

1. Select the cells from which you want to copy data.
2. Hover over the bottom-right corner of the selection to reveal the autofill icon.

3. Click and hold the autofill icon, then drag it to the target cells where you want to paste the copied data.
4. Release the mouse to complete the autofill action, and the data from the source cells will be copied and pasted into the target cells.

This feature is enabled by defining [enableAutoFill](#) property as **true**.

The following example demonstrates, how to enable autofill feature in the grid.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { SelectionSettingsModel, EditSettingsModel, ToolbarItems } from
 '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    FormsModule,
    FormsModule,

  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='272px'
[enableAutoFill]='true' [editSettings]='editSettings'
[toolbar]='toolbar' [selectionSettings]='selectionOptions'>
  <e-columns>
    <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
    isPrimaryKey='true' [visible]='false' width=120></e-
column>
    <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
    <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
    <e-column field='ShipCountry' headerText='Ship Name'
width=150></e-column>
    <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
  </e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public selectionOptions?: SelectionSettingsModel;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

```

        this.selectionOptions = { type: 'Multiple', cellSelectionMode:
'Box', mode: 'Cell' };
        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Batch' },
        this.toolbar = ['Add', 'Update', 'Cancel'];
    }
}

```

## MAIN.TS

```

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

```

\* If [enableAutoFill](#) is set to **true**, then the autofill icon will be displayed on cell selection to copy cells.

\* It requires the selection **mode** to be **Cell**, **cellSelectionMode** to be **Box** and also **editMode** to be **Batch**.

## Limitations

- AutoFill does not automatically convert string values to number or date types. If the selected cells contain string data and are dragged to number-type cells, the target cells will display **NaN**. Similarly, when dragging string-type cells to date-type cells, the target cells will display as an **empty cell**. It is important to ensure data types are compatible before using autofill to avoid unexpected results.
- The AutoFill feature does not support generating non-linear series or sequential data automatically. Cannot create complex series or patterns by simply dragging cells with non-sequential data. The autofill feature is designed for copying and pasting data from a selected range of cells.
- The Auto Fill feature can only be applied to the viewport cell when enabling the features of virtual scrolling, infinite scrolling, or column virtualization in the grid.

## Paste

The Syncfusion Angular Grid provides a paste feature that allows you to copy the content of a cell or a group of cells and paste it into another set of cells. This feature allows you to quickly copy and paste content within the grid, making it convenient for data entry and manipulation.

Follow the steps below to use the Paste feature in the grid:

1. Select the cells from which you want to copy the content.
2. Press the **Ctrl + C** shortcut key to copy the selected cells' content to the clipboard.
3. Select the target cells where you want to paste the copied content.
4. Press the **Ctrl + V** shortcut key to paste the copied content into the target cells.

## APP.COMPONENT.TS

```

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

```



```

import { GridModule, EditService, ToolbarService, SortService, PageService }
  from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { SelectionSettingsModel, EditSettingsModel, ToolbarItems } from
  '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' height='272px'
[editSettings]='editSettings' [toolbar]='toolbar'
[selectionSettings]='selectionOptions'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right'
isPrimaryKey='true' visible='false' width=90></e-
column>
      <e-column field='CustomerID' headerText='Customer ID'
width=100></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipCountry' headerText='Ship Name'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public selectionOptions?: SelectionSettingsModel;
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  ngOnInit(): void {
    this.data = data;
    this.selectionOptions = { type: 'Multiple', mode: 'Cell',
cellSelectionMode: 'Box' };
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Batch' },

```

```

        this.toolbar = ['Add', 'Update', 'Cancel'];
    }
}

```

## MAIN.TS

```

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

```

To perform paste functionality, it requires the selection **mode** to be **Cell**, **cellSelectionMode** to be **Box** and also Batch Editing should be enabled.

### Limitations

- The Paste feature does not automatically convert string values to number or date types. If the selected cells contain string data and are dragged to number-type cells, the target cells will display **NaN**. Similarly, when dragging string-type cells to date-type cells, the target cells will display as an **empty cell**. It is important to ensure data types are compatible before using AutoFill to avoid unexpected results.

## Context menu in Angular Grid component

The Syncfusion Angular Grid component comes equipped with a context menu feature, which is triggered when a user right-clicks anywhere within the grid. This feature serves to enrich the user experience by offering immediate access to a variety of supplementary actions and operations that can be executed on the data displayed in the grid.

To activate the context menu within the grid, you have an option to configure the grid's [contextMenuItems](#) property. You can set this property to either include the default context menu items or define your own custom context menu items, tailoring the menu options to suit your specific needs. This customization allows you to enhance the grid's functionality by providing context-sensitive actions for interacting with your data.

To use the context menu, you need to inject the **ContextMenuService** in the provider section of **AppModule**.

The context menu is triggered when you right-click on different areas of the grid, including:

- Header: When you right-click on the grid's header section.
- Content: When you right-click on the grid's main content area.
- Pager: When you right-click on the pager section.

The default context menu items in the header area of the grid are as follows:

Items	Description
-----	-----
<b>AutoFit</b>	Automatically adjust the column width to fit the content.
<b>AutoFitAll</b>	Automatically adjust all column widths to fit their content.

<b>Group</b>	Group the data based on the current column.	
<b>Ungroup</b>	Remove grouping for the current column.	
<b>SortAscending</b>	Sort the current column in ascending order.	
<b>SortDescending</b>	Sort the current column in descending order.	

The default context menu items in the content area of the grid are as follows:

Items	Description	
-----	-----	
<b>Edit</b>	Edit the currently selected record in the grid.	
<b>Delete</b>	Delete the currently selected record.	
<b>Save</b>	Save the changes made to the edited record.	
<b>Cancel</b>	Cancel the edit state and revert changes made to the edited record.	
<b>Copy</b>	Copy the selected records to the clipboard.	
<b>PdfExport</b>	Export the grid data as a PDF document.	
<b>ExcelExport</b>	Export the grid data as an Excel document.	
<b>CsvExport</b>	Export the grid data as a CSV document.	

The default context menu items in the pager area of the grid are as follows:

Items	Description	
-----	-----	
<b>FirstPage</b>	Navigate to the first page of the grid.	
<b>PrevPage</b>	Navigate to the previous page of the grid.	
<b>LastPage</b>	Navigate to the last page of the grid.	
<b>NextPage</b>	Navigate to the next page of the grid.	

The following example demonstrates how to enable context menu feature in the grid.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import {
  ContextMenuService, PageService, ResizeService, SortService,
  GroupService, EditService,
  PdfExportService, ExcelExportService
} from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ContextMenuItem, } from '@syncfusion/ej2-
angular-grids';
@Component({
  imports: [
```

```

        GridModule
    ],
    providers: [ContextMenuService,
        PageService,
        ResizeService,
        SortService,
        GroupService,
        EditService,
        PdfExportService,
        ExcelExportService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid [dataSource]='data' id="gridcomp"
allowPaging='true' [allowExcelExport]='true'
[allowPdfExport]='true' height='220px' [allowSorting]='true'
[allowGrouping]='true' [contextMenuItems]="contextMenuItems"
[editSettings]='editing'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID' width='90'
textAlign="Right" isPrimaryKey='true'></e-column>
            <e-column field='CustomerID' headerText='Customer Name'
width='100'></e-column>
            <e-column field='Freight' headerText='Freight' format='C2'
textAlign="Right" editType='numericedit' width='80'></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width='100'></e-column>
        </e-columns>
    </ejs-grid>`,
    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public contextMenuItems?: ContextMenuItem[];
        public editing?: EditSettingsModel;
        ngOnInit(): void {
            this.data = data;
            this.editing = { allowEditing: true, allowDeleting: true };
            this.contextMenuItems = ['AutoFit',
                'AutoFitAll',
                'SortAscending',
                'SortDescending',
                'Copy',
                'Edit',
                'Delete',
                'Save',
                'Cancel',
                'PdfExport',
                'ExcelExport',
                'CsvExport',
                'FirstPage',
                'PrevPage',
                'LastPage',
                'NextPage',
                'Group',
                'Ungroup'
            ];
        }
    }
}

```

```
}

```

### MAIN.TS

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

### Custom context menu items

The Syncfusion Angular Grid empowers you to enhance your user experience by incorporating custom context menu items into the default context menu. These customized options enable you to tailor the context menu to meet the unique requirements of your application.

To incorporate custom context menu items in the Syncfusion Angular Grid, you can achieve this by specifying the [contextMenuItems](#) property as a collection of [contextMenuItemModel](#). This allows you to define and customize the appearance and behavior of these additional context menu items according to your requirements.

Furthermore, you can assign actions to these custom items by utilizing the [contextMenuClick](#) event. This event provides you with the means to handle user interactions with the custom context menu items, enabling you to execute specific actions or operations when these items are clicked.

The following example demonstrates how to add custom context menu items in the Grid component.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, ContextMenuService, } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { employeeData } from './datasource';
import { GridComponent, ContextMenuItemModel } from '@syncfusion/ej2-angular-grids';
import { MenuEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [
    GridModule
  ],
  providers: [ContextMenuService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid id='grid' [dataSource]='data'
[allowSelection]='true'
[allowPaging]='true' height='265px'
[contextMenuItems]='contextMenuItems'
(contextMenuClick)='contextMenuClick($event)'>
    <e-columns>
      <e-column field='EmployeeID' [isPrimaryKey]='true'
headerText='Employee ID' textAlign='Right' width=120></e-column>
      <e-column field='FirstName' headerText='FirstName'
width=150></e-column>
```

```

        <e-column field='LastName' headerText='Last Name'
width=150></e-column>
        <e-column field='City' headerText='City'
width=150></e-column>
    </e-columns>
</ejs-grid>,
))
export class AppComponent implements OnInit {
    public data?: object[];
    public contextMenuItems?: ContextMenuItemModel[];
    @ViewChild('grid')
    public grid?: GridComponent;
    ngOnInit(): void {
        this.data = employeeData;
        this.contextMenuItems=[{ text: 'Copy with headers', target: '.e-
content', id: 'copywithheader' }];
    }
    contextMenuClick(args: MenuEventArgs): void {
        if (args.item.id === 'copywithheader') {
            (this.grid as GridComponent).copy(true);
        }
    }
}

```

### MAIN.TS

```

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

```

### Show context menu on left click

The Syncfusion Angular Grid provides the ability to show the context menu items on a left mouse click instead of the default right mouse click action.

This can be achieved by using the [created](#) event and the context menu's [beforeOpen](#) event of the Grid.

By using the [onclick](#) event listener of the Grid, you can obtain the clicked position values through the [ngAfterViewInit](#) method. This method is appropriate for interacting with the Document Object Model (DOM) and performing operations that require access to the rendered elements. The obtained positions are then sent to the [open](#) method of the context menu within the [onclick](#) event of the Grid. Additionally, the default action of right-clicking to open the context menu items is prevented by utilizing the [created](#) event of the Grid.

The following example demonstrates how to show context menu on left click using [created](#) event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ContextMenuService, PageService, SortService, ExcelExportService,
PdfExportService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';

```

```

import { GridComponent, ContextMenuItem, PageSettingsModel } from
 '@syncfusion/ej2-angular-grids';
import { BeforeOpenCloseMenuEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    GridModule

  ],
  providers: [ContextMenuService, PageService, SortService,
    ExcelExportService, PdfExportService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' #grid id="Grid"
[allowPaging]='true' [pageSettings]='pageSettings' [allowExcelExport]='true'
[allowPdfExport]='true'
[allowSorting]='true' [contextMenuItems]="contextMenuItems"
(created)="created()">
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
width='90' textAlign="Right" isPrimaryKey='true'></e-column>
      <e-column field='CustomerID' headerText='Customer
Name' width='100'></e-column>
      <e-column field='ShipCountry' headerText='Ship
Country' width='100'></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width='100'></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public values?: any;
  public data?: object[];
  public contextMenuItems?: ContextMenuItem[];
  public pageSettings?: PageSettingsModel;
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
    this.contextMenuItems = [
      'SortAscending',
      'SortDescending',
      'FirstPage',
      'PrevPage',
      'LastPage',
      'NextPage',
      'PdfExport',
      'ExcelExport',
    ];
    this.pageSettings = { pageSize: 8 };
  }
  ngAfterViewInit(): void {
    const gridElement = document.getElementById('Grid');
    (gridElement as HTMLElement).onclick = (event: MouseEvent) => {
      this.values = event;
      (this.grid as
GridComponent).contextMenuModule.contextMenu.open(
        this.values.pageY + pageYOffset,

```

```

        this.values.pageX + pageXOffset
    );
}

}

created(): void {
    (this.grid as
GridComponent).contextMenuModule.contextMenu.beforeOpen = (
    args: BeforeOpenCloseMenuEventArgs
    ) => {
        if (args.event instanceof MouseEvent && args.event.which === 3)
        {
            args.cancel = true;
        }
        args.event = this.values;
        (this.grid as any).contextMenuModule.contextMenuBeforeOpen(
            args
        );
    };
}
}
}

```

## 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 hide or show an item in context menu for specific area inside of grid by defining the [target](#) property.

### Enable or disable context menu items

With the Syncfusion Angular Grid, you have the ability to manage the activation or deactivation of both default and custom context menu items. This feature provides you with the flexibility to tailor the behavior of context menu items to suit specific conditions or individual interactions within your application.

This can be achieved using the [enableItems](#) method of the context menu. By setting the enable parameter in the enableItems method to **true**, you can enable context menu items, and by setting it to **false**, you can disable them. Based on your specific condition or requirements, you can enable or disable the context menu item using the `enableItems` method.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to enable and disable the context menu items using `enableItems` method. When the switch is toggled, the [change](#) event is triggered, and the **Copy** items is updated accordingly.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ContextMenuService, PageService, EditService } from
 '@syncfusion/ej2-angular-grids'

```



```

import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, EditSettingsModel, ContextMenuItem } from '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
    imports: [

        GridModule,
        ButtonModule,
        CheckBoxModule,
        RadioButtonModule,
        SwitchModule,
    ],
    providers: [ContextMenuService, PageService, EditService],
    standalone: true,
    selector: 'app-root',
    template: `
<div>
<label style="padding: 10px 10px">
Enable or disable context menu items
</label>
<ejs-switch id="switch" (change)="switchChange($event)"></ejs-switch>
</div>
<ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
id="grid" allowPaging='true' height='265px'
[contextMenuItems]="contextMenuItems" [editSettings]='editing' >
    <e-columns>
        <e-column field='OrderID' headerText='Order ID'
width='90' textAlign="Right" isPrimaryKey='true'></e-column>
        <e-column field='CustomerID' headerText='Customer Name'
width='100'></e-column>
        <e-column field='Freight' headerText='Freight'
format='C2' textAlign="Right" editType='numericedit' width='90'></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width='150'></e-column>
    </e-columns>
</ejs-grid>`,
    })
export class AppComponent implements OnInit {
    public data?: object[];
    public contextMenuItems?: ContextMenuItem[];
    @ViewChild('grid')
    public grid?: GridComponent;
    public editing?: EditSettingsModel;
    ngOnInit(): void {
        this.data = data;
        this.contextMenuItems = ['Copy', 'Edit', 'Delete'];
        this.editing = { allowAdding: true, allowDeleting: true,
allowEditing: true };
    }
}

```

```

switchChange(args: ChangeEventArgs) {
    if (args.checked) {
        (this.grid as
GridComponent).contextMenuModule.contextMenu.enableItems(['Copy'], false);
    } else {
        (this.grid as
GridComponent).contextMenuModule.contextMenu.enableItems(['Copy'], true);
    }
}
}

```

## 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 context menu items

The Syncfusion Angular Grid provides the flexibility to show or hide both default and custom context menu items. This feature allows you to customize the context menu items based on various conditions or individuals interactions.

This can be achieved using the [showItems](#) and [hideItems](#) methods of the context menu by specifying the item you want to show or hide as an argument.

In the following example, the [EJ2 Toggle Switch Button](#) component is added to show or hide the context menu items using [showItems](#) and [hideItems](#) methods. When the switch is toggled, the [change](#) event is triggered, and the **Edit** and **Delete** items are updated accordingly.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { ContextMenuService, PageService, EditService } from
 '@syncfusion/ej2-angular-grids'
import {
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,
    SwitchModule,
} from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent, EditSettingsModel, ContextMenuItem } from
 '@syncfusion/ej2-angular-grids';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-buttons';
@Component({
  imports: [

    GridModule,
    ButtonModule,
    CheckBoxModule,
    RadioButtonModule,

```

```

        SwitchModule,
    ],
    providers: [ContextMenuService, PageService, EditService],
    standalone: true,
    selector: 'app-root',
    template: `
        <div>
            <label style="padding: 10px 10px">
                Show or hide context menu items
            </label>
            <ejs-switch id="switch" (change)="switchChange($event)"></ejs-switch>
        </div>
        <ejs-grid #grid style="padding: 10px 10px" [dataSource]='data'
            id="grid" allowPaging='true' height='265px'
                [contextMenuItems]="contextMenuItems" [editSettings]='editing' >
                    <e-columns>
                        <e-column field='OrderID' headerText='Order ID'
                            width='90' textAlign="Right" isPrimaryKey='true'></e-column>
                        <e-column field='CustomerID' headerText='Customer Name'
                            width='100'></e-column>
                        <e-column field='Freight' headerText='Freight'
                            format='C2' textAlign="Right" editType='numericedit' width='90'></e-column>
                        <e-column field='ShipCity' headerText='Ship City'
                            width='150'></e-column>
                    </e-columns>
                </ejs-grid>`,
    ))
export class AppComponent implements OnInit {
    public data?: object[];
    public contextMenuItems?: ContextMenuItem[];
    @ViewChild('grid')
    public grid?: GridComponent;
    public editing?: EditSettingsModel;
    ngOnInit(): void {
        this.data = data;
        this.contextMenuItems = ['Copy', 'Edit', 'Delete'];
        this.editing = { allowAdding: true, allowDeleting: true, allowEditing:
true };
    }
    switchChange(args: ChangeEventArgs) {
        if (args.checked) {
            (this.grid as GridComponent).contextMenuModule.contextMenu.hideItems([
                'Edit Record',
                'Delete Record',
            ]);
        } else {
            (this.grid as GridComponent).contextMenuModule.contextMenu.showItems([
                'Edit Record',
                'Delete Record',
            ]);
        }
    }
}

```

## 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 Grid component

The Grid component offers various ways to customize its appearance using both default CSS and custom themes. Let's go over some common approaches:

### Default CSS overrides:

You can use custom CSS to override the default styles of the Grid component. This allows you to change colors, fonts, paddings, and more. You can inspect the generated HTML of the Grid using browser developer tools to identify the relevant CSS classes and styles.

Here's a basic example of how you can override the header background color of the Grid:

```
`css
```

```
/ In your component's CSS file /
```

```
.e-grid .e-headercell {
```

```
background-color: #333; / Override the header background color /
```

```
color: #fff;
```

```
}
```

```
,
```

ORDER ID ▼	CUSTOMER ID ▼	FREIGHT ▼	ORDER DATE ▼
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996
10251	VICTE	\$41.34	7/8/1996
10252	SUPRD	\$51.30	7/9/1996
10253	HANAR	\$58.17	7/10/1996

### Using theme studio:

Syncfusion's Theme Studio tool allows you to create custom themes for all their controls, including the Grid. This is a more advanced approach that lets you define a comprehensive set of styles to achieve a consistent look and feel throughout your application.

1. Visit the [Syncfusion Theme Studio](#).
2. Select the Grid control from the left panel.
3. Customize various aspects of the control's appearance, such as colors, typography, and spacing.

- Once done, you can download the generated CSS file and include it in your Angular project.

#### Customizing the grid root element

To customize the appearance of the root element of the Syncfusion Angular Grid component, you can use CSS. Here's an example of how to modify the font family and row colors using CSS:

```
`css
.e-grid {
font-family: cursive;
}
```

ORDER ID ▼	CUSTOMER ID ▼	FREIGHT ▼	ORDER DATE ▼
10248	VINET	\$32.38	7/4/1996
10249	TOMSP	\$11.61	7/5/1996
10250	HANAR	\$65.83	7/8/1996
10251	VICTE	\$41.34	7/8/1996
10252	SUPRD	\$51.30	7/9/1996
10253	HANAR	\$58.17	7/10/1996

The above code snippet, the **.e-grid** class targets the root element of the Syncfusion Angular Grid component, and the **font-family** property is set to **cursive** to change the font family of the grid content.

In the following sample, the font family of grid content is changed to **cursive**, and the background color of rows, selected rows, alternate rows, and row hovering color is modified using the below CSS styles.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { PageService, GridComponent, SelectionSettingsModel } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowPaging]='true'
[pageSettings]='pageSettings' [selectionSettings]='selectionOptions'
height='268px'>`
})
```

```

        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
type='number' isPrimaryKey='true' textAlign='Right' width=100></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
type='string' width=120></e-column>
            <e-column field='Freight' headerText='Freight'
type='number' textAlign='Right' format='C' width=100></e-column>
            <e-column field='ShipName' headerText='Ship Name'
type='string' width=180></e-column>
        </e-columns>
    </ejs-grid>,
    providers: [PageService]
})
export class AppComponent implements OnInit {
    public data?: Object[];
    public grid?: GridComponent;
    public pageSettings?: Object;
    public selectionOptions?: SelectionSettingsModel;
    ngOnInit(): void {
        this.data = data;
        this.pageSettings = { pageSize: 8 };
        this.selectionOptions = { type: 'Multiple' };
    }
}

```

## INDEX.HTML

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Syncfusion Angular Grid</title>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta name="description" content="Typescript UI Controls" />

    <meta name="author" content="Syncfusion" />
    <style>
        .e-grid {
            font-family: cursive;
        }
        .e-grid .e-row:hover .e-rowcell {
            background-color: rgb(204, 229, 255) !important;
        }
        .e-grid .e-rowcell.e-selectionbackground {
            background-color: rgb(230, 230, 250);
        }
        .e-grid .e-row.e-altrow {
            background-color: rgb(150, 212, 212);
        }
        .e-grid .e-row {
            background-color: rgb(180, 180, 180);
        }
    </style>
</head>
<body style="margin-top: 125px">
    <app-root>

```

```

        <div id='loader'>Loading....</div>
    </app-root>
</body>
</html>

```

## 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

- [Footer template styling in Angular Grid](#)

## Ej1 api migration in Angular Grid component

This article describes the API migration process of Grid component from Essential JS 1 to Essential JS 2.

### Sorting

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowSorting* <br><br><ej-grid [allowSorting]="true"><br></ej-grid>| **Property:** *allowSorting* <br><br><ejs-grid [allowSorting]="true"><br></ejs-grid>|

| Clear the Sorted columns | **Method:** *clearSorting()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.clearSorting();<br>&#160;}<br>}

**Method:** *clearSorting()*<br><br>@ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.clearSorting()

| Get the Sorted Columns by using the Fieldname | **Method:** *getsortColumnByField(field)*  
<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any,  
any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getsortColumnByField("OrderID");<br>&#160;}<br>} | **Property:** *sortSettings.columns*<br><br>You can get a sorted column by iterating  
*sortSettings.columns* with fieldname

| Remove the Sorted Columns | **Method:** *removeSortedColumns(fieldName)* <br><br>export class  
AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.removeSortedColumns("OrderID");<br>&#160;}<br>} | **Method:** *removeSortColumn(fieldName)*<br><br>@ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.removeSortColumn("OrderID")

| Sort a Column by using the method | **Method:** *sortColumn(columnName, [sortingDirection])*  
<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any,  
any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.sortColumn("OrderID",

"ascending");<br> &#160;}<br>} | **Method:** *sortColumn(columnName, Direction)*<br><br>  
@ViewChild('grid') Grid: GridComponent;<br>this.Grid.sortColumn("OrderID", "ascending")

### Grouping

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowGrouping* <br><br><ej-grid [allowGrouping]="true"><br></ej-grid>  
**Property:** *allowGrouping* <br><br><ejs-grid [allowGrouping]="true"><br></ejs-grid>|

| Group Columns initially | **Property:** *groupSettings.groupedColumns* <br><br><ej-grid  
[allowGrouping]="true" [groupSettings]="groupOptions"><br></ej-grid><br> **TS**  
<br>this.groupOptions = {groupedColumns:["OrderID"]}; | **Property:** *groupSettings.columns*  
<br><br><ejs-grid [allowGrouping]="true" [groupSettings]="groupOptions"><br></ejs-  
grid><br> **TS**<br>this.groupOptions = {columns:["OrderID"]};|

| Caption Template | **Property:** *groupSettings.captionFormat* <br><br><ej-grid  
[allowGrouping]="true" [groupSettings]="groupOptions"><br></ej-grid><br> **TS**  
<br>this.groupOptions = {captionFormat: "#template"}; | **Property:** *groupSettings.captionTemplate*  
<br><br><ejs-grid [allowGrouping]="true" [groupSettings]="groupOptions"><br></ejs-  
grid><br> **TS**<br>this.groupOptions = {captionTemplate: "#template"};|

| Show Drop Area | **Property:** *groupSettings.showDropArea* <br><br><ej-grid [allowGrouping]="true"  
[groupSettings]="groupOptions"><br></ej-grid><br> **TS** <br>this.groupOptions =  
{showDropArea:false}; | **Property:** *groupSettings.showDropArea* <br><br><ejs-grid  
[allowGrouping]="true" [groupSettings]="groupOptions"><br></ejs-  
grid><br> **TS**<br>this.groupOptions = {showDropArea:false};|

| Collapse all group caption rows | **Method:** *collapseAll()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.collapseAll();<br> &#160;}<br>} |  
**Method:** *collapseAll()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.collapseAll()

| Expand all group caption rows | **Method:** *expandAll()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.expandAll();<br> &#160;}<br>} |  
**Method:** *expandAll()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.expandAll()

| Expand or collapse the row based <br>on the row state in grid | **Method:** *expandCollapse(\$target)*  
<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any,  
any>;<br> &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.expandCollapse(\$(".tr td.e-  
recordplusexand > div").first());<br> &#160;}<br>} | **Method:** *expandCollapseRows()*<br><br>  
@ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.groupModule.expandCollapseRows(<br>gridObj.getContent().que  
rySelectorAll('.e-recordplusexand')[0]))



| Collapse the group drop area in grid | **Method:** *collapseGroupDropArea()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.collapseGroupDropArea();<br>&#160;}<br>} | Not Applicable

| Expand the group drop area in grid | **Method:** *expandGroupDropArea()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.expandGroupDropArea();<br>&#160;}<br>} | Not Applicable

| Group a column by using the method | **Method:** *groupByColumn(fieldName)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.groupColumn("OrderID");<br>&#160;}<br>} | **Method:** *groupByColumn(fieldName)*<br><br>@ViewChild('grid') Grid: GridComponent;<br>this.Grid.groupColumn("OrderID")

| Ungroup a grouped column by using the method | **Method:** *ungroupColumn(fieldName)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.ungroupColumn("OrderID");<br>&#160;}<br>} | **Method:** *ungroupColumn(fieldName)*<br><br>@ViewChild('grid') Grid: GridComponent;<br>this.Grid.ungroupColumn("OrderID")

## Filtering

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowFiltering* <br><br><ej-grid [allowFiltering]="true"><br></ej-grid> | **Property:** *allowFiltering* <br><br><ejs-grid [allowFiltering]="true"><br></ejs-grid> |

| Menu Filtering | **Property:** *filterSettings.filterType* <br><br><ej-grid [allowFiltering]="true" [filterSettings]="filterOptions"><br></ej-grid><br> **TS** <br>this.filterOptions = { filterType : "menu" }; | **Property:** *filterSettings.type* <br><br><ejs-grid [allowFiltering]="true" [filterSettings]="filterOptions"><br></ejs-grid><br> **TS** <br>this.filterOptions = { type:'Menu' }; |

| Excel Filtering | **Property:** *filterSettings.filterType* <br><br><ej-grid [allowFiltering]="true" [filterSettings]="filterOptions"><br></ej-grid><br> **TS** <br>this.filterOptions = { filterType : "excel" }; | **Property:** *filterSettings.type* <br><br><ejs-grid [allowFiltering]="true" [filterSettings]="filterOptions"><br></ejs-grid><br> **TS** <br>this.filterOptions = { type:'Excel' }; |

| Clear the Filtered values | **Method:** *clearFiltering(field)* - *field is optional* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.clearFiltering();<br>&#160;}<br>} | **Method:** *clearFiltering()*<br><br>@ViewChild('grid') Grid: GridComponent;<br>this.Grid.clearFiltering()

| Filter a column by using the method | **Method:** *filterColumn(fieldName, filterOperator, filterValue, predicate, [matchcase],[actualFilterValue])* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.filterColumn("OrderID", "contains", "123", true, false, "123");<br>&#160;}<br>} |

```
&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.filterColumn("OrderID",<br>"equal",
"10248","and", true);<br>&#160;}<br>} | Method: filterByColumn(fieldName, filterOperator,
filterValue, predicate, matchCase, ignoreAccent, actualFilterValue, actualOperator)<br><br>
```

```
@ViewChild('grid') Grid:
```

```
GridComponent;<br>this.Grid.filterByColumn("OrderID","equal",10248)
```

```
| Filter columns by Collection | Method: filterColumn(filterCollection) <br><br>export class
AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.filterColumn({field:"OrderID",<br>&#160;operator:"lessthan",value:"10266",<br>predicate:"and",matchcase:true},<br>&#160;{field:
"EmployeeID",operator:<br>&#160;"equal",value:2,predicate:"and", matchcase:true}});<br>
&#160;}<br>} | Property: filterSettings.columns <br><br><ej-grid allowFiltering="true"
[filterSettings]="filterOptions"><br></ej-grid><br>TS<br>this.filterOptions = { columns: [{ field:
'ShipCity', matchCase: false, operator: 'startswith', predicate: 'and', value: 'reims' },<br>{ field:
'ShipCountry', matchCase: false, operator: 'startswith', predicate: 'and', value: 'France' } ] };|
```

```
| Get the Filtered Records | Method: getFilteredRecords() <br><br>export class AppComponent
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getFilteredRecords();<br>
&#160;}<br>} | Not Applicable
```

## Searching

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
|-----|-----|-----|
```

```
| Default | Property: toolbarSettings.toolbarItems <br><br><ej-grid [allowSearching]="true"
[toolbarSettings]="toolbarOptions"><br></ej-grid><br>TS <br>this.toolbarOptions =
{showToolbar:true,toolbarItems:["search"]};| Property: toolbar <br><br><ej-grid
[toolbar]="toolbar"><br></ej-grid><br>TS<br>this.toolbar = ['Search'];|
```

```
| Clear the Searched values | Method: clearSearching() <br><br>export class AppComponent
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.clearSearching();<br>&#160;}<br>} |
Method: searchModule.search()<br><br>@ViewChild('grid') Grid:
GridComponent;<br>this.Grid.searchModule.search("");
```

```
| Search a value | Method: search(searchString) <br><br>export class AppComponent
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.search("France");<br>&#160;}<br>} |
Method: searchModule.search()<br><br>@ViewChild('grid') Grid:
GridComponent;<br>this.Grid.searchModule.search("France");
```

## Paging

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
|-----|-----|-----|
```

```
| Default | Property: allowPaging <br><br><ej-grid [allowPaging]="true"><br></ej-grid>| Property:
allowPaging <br><br><ej-grid [allowPaging]="true"><br></ej-grid>|
```

| Customize Paging | **Property:** *pageSettings.pageSize* `<br><br><ej-grid [allowPaging]="true" [pageSettings]="pageOptions"><br></ej-grid><br> TS <br>this.pageOptions = {pageSize: 5};|`  
**Property:** *pageSettings.pageSize* `<br><br><ejs-grid [allowPaging]="true"[pageSettings]="pageOptions"><br></ejs-grid><br>TS<br>this.pageOptions = {pageSize: 5,pageSizes:["10, 15"]}|`

| Change Page Size | **Method:** *changePageSize(pageSize)* `<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.changePageSize(7);<br>&#160;}<br>}`  
**Property:** *pageSettings.pageSize* `<br><br>Pagesize can be modified by using the below code<br><ej-grid [allowPaging]="true"[pageSettings]="pageOptions"><br></ejs-grid><br>TS<br>this.pageOptions = {pageSize: 7;}|`

| Get Current Page Index | **Method:** *getCurrentIndex()* `<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getCurrentIndex();<br>&#160;}<br>}`  
**Property:** *pageSettings.currentPage* `<br><br><ejs-grid [allowPaging]="true"><br></ejs-grid><br>TS<br>var currentPage: any = this.pageSettings.currentPage;|`

| Get Pager Element | **Method:** *getPager()* `<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getPager();<br>&#160;}<br>}|`  
**Method:** *getPager()* `<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getPager();`

| Send a paging request to the specified Page | **Method:** *gotoPage(pageIndex)* `<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.gotoPage(3);<br>&#160;}<br>}|`  
**Method:** *gotoPage(pageIndex)* `<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.gotoPage(3);`

| Calculate Pagesize of grid by using its Parent height(containerHeight) | **Method:** *calculatePageSizeByParentHeight(containerHeight)* `<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget<br>.calculatePageSizeByParentHeight(400);<br>&#160;}<br>}|` Not Applicable

## Selection

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowSelection* `<br><br><ej-grid [allowSelection]="true"><br></ej-grid>|`  
**Property:** *allowSelection* `<br><br><ejs-grid [allowSelection]="true"><br></ejs-grid>|`

| Single Selection | **Property:** *selectionType* `<br><br><ej-grid [allowSelection]="true" selectionType="single"><br></ej-grid>|` **Property:** *selectionSettings.type* `<br><br><ejs-grid [allowSelection]="true" [selectionSettings]="selectOptions"><br></ejs-grid><br>TS<br>this.selectOptions = { type: 'Single' };|`

| Multiple Selection | **Property:** *selectionType* <br><br><ej-grid [allowSelection]="true" selectionType="multiple"></ej-grid> | **Property:** *selectionSettings.type* <br><br><ejs-grid [allowSelection]="true" [selectionSettings]="selectOptions"><br></ejs-grid><br>**TS**<br>this.selectOptions = { type: 'Multiple' };|

| Row Selection | **Property:** *selectionSettings.selectionMode* <br><br><ej-grid [allowSelection]="true" [selectionSettings]="selectionMode"></ej-grid><br> **TS** <br>this.selectionMode = {selectionMode :["row"]};| **Property:** *selectionSettings.mode* <br><br><ejs-grid [allowSelection]="true" [selectionSettings]="selectOptions"><br></ejs-grid><br>**TS**<br>this.selectOptions = { mode: 'Row' };|

| Cell Selection | **Property:** *selectionSettings.selectionMode* <br><br><ej-grid [allowSelection]="true" [selectionSettings]="selectionMode"></ej-grid><br> **TS** <br>this.selectionMode = {selectionMode :["cell"]};| **Property:** *selectionSettings.mode* <br><br><ejs-grid [allowSelection]="true" [selectionSettings]="selectOptions"><br></ejs-grid><br>**TS**<br>this.selectOptions = { mode: 'Cell' };|

| Clear the selected Cells | **Method:** *clearCellSelection()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.clearCellSelection();<br>&#160;}<br>} | **Method:** *clearCellSelection()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.selectionModule.clearCellSelection()

| Clear the selected Columns | **Method:** *clearColumnSelection([index])* - index is optional <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.clearColumnSelection();<br>&#160;}<br>} | Not Applicable

| Get the selected Records | **Method:** *getSelectedRecords()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getSelectedRecords();<br>&#160;}<br>} | **Method:** *getSelectedRecords()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getSelectedRecords()

| Get the selected Rows | **Method:** *getSelectedRows()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getSelectedRows();<br>&#160;}<br>} | **Method:** *getSelectedRows()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getSelectedRows()

| Select Cells | **Method:** *selectCells(rowCellIndexes)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.selectCells([[1, [4, 3, 2]]]);<br>&#160;}<br>} | **Method:** *selectionModule.selectCells()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.selectionModule.selectCells([{ rowIndex: 0, cellIndexes: [0] }, { rowIndex: 1, cellIndexes: [1] }]);

| Select Rows | **Method:** *selectRows(fromIndex, toIndex)* <br><br>export class AppComponent  
 {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.selectRows(1, 4);<br> &#160;}<br>} |  
**Method:** *selectionModule.selectRows(rowIndexes)*<br><br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.selectionModule.selectRows([0, 2])

| Triggers when a <br>cell is selected | **Event:** *cellSelected* <br><br><ej-grid #grid (cellSelected) =  
 "cellSelected(\$event)"><br></ej-grid><br> **TS** <br>cellSelected(e: any){} | **Event:** *cellSelected*  
 <br><br><ejs-grid (cellSelected)='cellSelected(\$event)'><br></ejs-grid><br> **TS**  
 <br>cellSelected(args: any): void{}

| Triggers before the cell is being selected | **Event:** *cellSelecting* <br><br><ej-grid #grid (cellSelecting) =  
 "cellSelecting(\$event)"><br></ej-grid><br> **TS** <br>cellSelecting(e: any){} | **Event:** *cellSelecting*  
 <br><br><ejs-grid (cellSelecting)='cellSelecting(\$event)'><br></ejs-grid><br> **TS**  
 <br>cellSelecting(args: any): void{}

| Triggers when a <br>cell is deselected | **Event:** *cellDeselected* <br><br><ej-grid #grid (cellDeselected)  
 = "cellDeselected(\$event)"><br></ej-grid><br> **TS** <br>cellDeselected(e: any){} | **Event:**  
*cellDeselected* <br><br><ejs-grid (cellDeselected)='cellDeselected(\$event)'><br></ejs-grid><br> **TS**  
 <br>cellDeselected(args: any): void{}

| Triggers before the cell is being deselected | **Event:** *cellDeselecting* <br><br><ej-grid #grid  
 (cellDeselecting) = "cellDeselecting(\$event)"><br></ej-grid><br> **TS** <br>cellDeselecting(e:  
 any){} | **Event:** *cellDeselecting* <br><br><ejs-grid  
 (cellDeselecting)='cellDeselecting(\$event)'><br></ejs-grid><br> **TS** <br>cellDeselecting(args:  
 any): void{}

| Triggers when the <br>row is selected | **Event:** *rowSelected* <br><br><ej-grid #grid (rowSelected) =  
 "rowSelected(\$event)"><br></ej-grid><br> **TS** <br>rowSelected(e: any){} | **Event:** *rowSelected*  
 <br><br><ejs-grid (rowSelected)='rowSelected(\$event)'><br></ejs-grid><br> **TS**  
 <br>rowSelected(args: any): void{}

| Triggers before the row is being selected | **Event:** *rowSelecting* <br><br><ej-grid #grid (rowSelecting)  
 = "rowSelecting(\$event)"><br></ej-grid><br> **TS** <br>rowSelecting(e: any){} | **Event:** *rowSelecting*  
 <br><br><ejs-grid (rowSelecting)='rowSelecting(\$event)'><br></ejs-grid><br> **TS**  
 <br>rowSelecting(args: any): void{}

| Triggers when the <br>row is deselected | **Event:** *rowDeselected* <br><br><ej-grid #grid  
 (rowDeselected) = "rowDeselected(\$event)"><br></ej-grid><br> **TS** <br>rowDeselected(e:  
 any){} | **Event:** *rowDeselected* <br><br><ejs-grid  
 (rowDeselected)='rowDeselected(\$event)'><br></ejs-grid><br> **TS** <br>rowDeselected(args:  
 any): void{}

| Triggers before the row is being deselected | **Event:** *rowDeselecting* <br><br><ej-grid #grid  
 (rowDeselecting) = "rowDeselecting(\$event)"><br></ej-grid><br> **TS** <br>rowDeselecting(e:  
 any){} | **Event:** *rowDeselecting* <br><br><ejs-grid  
 (rowDeselecting)='rowDeselecting(\$event)'><br></ejs-grid><br> **TS** <br>rowDeselecting(args:  
 any): void{}

| Triggers when the column is selected | **Event:** *columnSelected* <br><br><ej-grid #grid  
(columnSelected) = "columnSelected(\$event)"><br></ej-grid><br> **TS** <br>columnSelected(e:  
any){}| Not Applicable

| Triggers before the column is being selected | **Event:** *columnSelecting* <br><br><ej-grid #grid  
(columnSelecting) = "columnSelecting(\$event)"><br></ej-grid><br> **TS** <br>columnSelecting(e:  
any){}| Not Applicable

| Triggers when the column is deselected | **Event:** *columnDeselected* <br><br><ej-grid #grid  
(columnDeselected) = "columnDeselected(\$event)"><br></ej-grid><br> **TS**  
<br>columnDeselected(e: any){}| Not Applicable

| Triggers before the column is being deselected | **Event:** *columnDeselecting* <br><br><ej-grid #grid  
(columnDeselecting) = "columnDeselecting(\$event)"><br></ej-grid><br> **TS**  
<br>columnDeselecting(e: any){}| Not Applicable

## Editing

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *editSettings* <br><br><ej-grid [editSettings]="editSettings"><br></ej-grid><br>  
**TS** <br>this.editSettings = {allowEditing: true, allowAdding: true, allowDeleting: true}; | **Property:**  
*editSettings* <br><br><ejs-grid [editSettings]="editSettings"><br></ejs-  
grid><br>**TS** <br>this.editSettings = { allowEditing: true, allowAdding: true, allowDeleting: true }; |

| Inline Editing | **Property:** *editSettings.editMode* <br><br><ej-grid  
[editSettings]="editSettings"><br></ej-grid><br> **TS** <br>this.editSettings = {allowEditing: true,  
allowAdding: true, allowDeleting: true, editMode : "normal"}; | **Property:** *editSettings.mode*  
<br><br><ejs-grid [editSettings]="editSettings"><br></ejs-grid><br> **TS** <br>this.editSettings = {  
allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Normal' }; |

| Dialog Editing | **Property:** *editSettings.editMode* <br><br><ej-grid  
[editSettings]="editSettings"><br></ej-grid><br> **TS** <br>this.editSettings = {allowEditing: true,  
allowAdding: true, allowDeleting: true, editMode : "dialog"}; | **Property:** *editSettings.mode*  
<br><br><ejs-grid [editSettings]="editSettings"><br></ejs-grid><br> **TS** <br>this.editSettings = {  
allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Dialog' }; |

| Batch Editing | **Property:** *editSettings.editMode* <br><br><ej-grid  
[editSettings]="editSettings"><br></ej-grid><br> **TS** <br>this.editSettings = {allowEditing: true,  
allowAdding: true, allowDeleting: true, editMode : "batch"}; | **Property:** *editSettings.mode*  
<br><br><ejs-grid [editSettings]="editSettings"><br></ejs-grid><br> **TS** <br>this.editSettings = {  
allowEditing: true, allowAdding: true, allowDeleting: true, mode: 'Batch' }; |

| Add a new Record | **Method:** *addRecord([data,[serverChange]])* <br><br>export class  
AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
&#160;ngAfterViewInit(){<br>&#160;&#160;&#160;this.Grid.widget.addRecord({{"OrderID":12333});<br>  
&#160;}<br>} | **Method:** *addRecord(data(optional), index(optional))*<br><br>@ViewChild('grid')  
Grid: GridComponent;<br>this.Grid.addRecord({OrderID:10000});



| Batch Cancel | **Method:** *batchCancel()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.batchCancel();<br> &#160;}<br>} |  
Not Applicable

| Batch Save | **Method:** *batchSave()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.batchSave();<br> &#160;}<br>} |  
**Method:** *batchSave()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.editModule.batchSave();

| Save a Cell - If *preventSaveEvent* is true, then it <br>will prevent the client side cellSave event |  
**Method:** *saveCell([preventSaveEvent])* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.saveCell();<br> &#160;}<br>} |  
**Method:** *saveCell()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.editModule.saveCell();

| End Edit | **Method:** *endEdit()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid')  
Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.endEdit();<br> &#160;}<br>} |  
**Method:** *endEdit()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.endEdit();

| Cancel Edit | **Method:** *cancelEdit()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.cancelEdit();<br> &#160;}<br>} |  
**Method:** *closeEdit()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.closeEdit();

| Delete Record | **Method:** *deleteRecord(fieldName, data)* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.deleteRecord("OrderID", { OrderID:  
10249, EmployeeID: 3 });<br> &#160;}<br>} | **Method:** *deleteRecord(field, data)*<br><br>  
@ViewChild('grid') Grid: GridComponent;<br>this.Grid.deleteRecord("OrderID", { OrderID:  
10249, EmployeeID: 3 });

| Delete Row | **Method:** *deleteRow(\$tr)* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.deleteRow(\$(".gridcontent  
tr").first());<br> &#160;}<br>} | **Method:** *deleteRow(tr)*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.deleteRow(this.Grid.getContentTable());

| Edit a cell in Batch edit mode | **Method:** *editModule.editCell(index, fieldName)* <br><br>export class  
AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.editCell(2, "OrderID");<br> &#160;}<br>} | **Method:** *editCell(index, field)*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.editModule.editCell(0, "CustomerID")

| Edit Form Validation | **Method:** *editFormValidate()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>

```

    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.editFormValidate();<br> &#160;}<br>
    | Method: editModule.formObj.validate()<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.editModule.formObj.validate()

    | Get Batch Changes| Method: getBatchChanges() <br><br>export class AppComponent
    {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getBatchChanges();<br> &#160;}<br>
    | Method: editModule.getBatchChanges()<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.editModule.getBatchChanges()

    | Refresh Batch Edit Changes| Method: refreshBatchEditChanges() <br><br>export class
    AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshBatchEditChanges();<br>
    &#160;}<br>} | Not Applicable

    | Set Default Data for adding| Method: setDefaultData(defaultData) <br><br>export class
    AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.setDefaultData({OrderID:10000});<br>
    > &#160;}<br>} | Method: editModule.addRecord()<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.editModule.addRecord({OrderID:10000});

    | Start Edit| Method: startEdit($tr) <br><br>export class AppComponent
    {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.startEdit($(".gridcontent
    tr").first());<br> &#160;}<br>} | Method: startEdit()<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.startEdit(this.Grid.selectRow(0))

    | Update Record| Method: updateRecord(fieldName, data) <br><br>export class AppComponent
    {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.updateRecord("OrderID", { OrderID:
    10249, EmployeeID: 3 });<br> &#160;}<br>} | Not Applicable

    | Get Currently edited cell data| Method: getCurrentEditCellData() <br><br>export class
    AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getCurrentEditCellData();<br>
    &#160;}<br>} | Method: editModule.getCurrentEditCellData()<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.editModule.getCurrentEditCellData();

    | Set Cell value| Method: setCellText(rowIndex, cellIndex, value) <br><br>export class AppComponent
    {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>
    &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.setCellText(0, 1, "France");<br>
    &#160;}<br>} | Method: setCellValue(key, field, value)<br><br> @ViewChild('grid') Grid:
    GridComponent;<br>this.Grid.setCellValue(10248,"CustomerID","A");

    | Triggers when adding a <br>record in batch editing| Event: batchAdd <br><br><ej-grid #grid
    (batchAdd) = "batchAdd($event)"><br></ej-grid><br> TS <br>batchAdd(e: any){} | Event:
    batchAdd <br><br><ejs-grid (batchAdd)='batchAdd($event)'><br></ejs-grid><br> TS
    <br>batchAdd(args: any): void{}
  
```



| Triggers when deleting a <br>record in batch editing| **Event:** *batchDelete* <br><br><ej-grid #grid (batchDelete) = "batchDelete(\$event)"><br></ej-grid><br> **TS** <br>batchDelete(e: any){}| **Event:** *batchDelete* <br><br><ejs-grid (batchDelete)= 'batchDelete(\$event)'><br></ejs-grid><br> **TS** <br>batchDelete(args: any): void{}

| Triggers before adding a record in batch editing| **Event:** *beforeBatchAdd* <br><br><ej-grid #grid (beforeBatchAdd) = "beforeBatchAdd(\$event)"><br></ej-grid><br> **TS** <br>beforeBatchAdd(e: any){}| **Event:** *beforeBatchAdd* <br><br><ejs-grid (beforeBatchAdd)= 'beforeBatchAdd(\$event)'><br></ejs-grid><br> **TS** <br>beforeBatchAdd(args: any): void{}

| Triggers before deleting a record in batch editing| **Event:** *beforeBatchDelete* <br><br><ej-grid #grid (beforeBatchDelete) = "beforeBatchDelete(\$event)"><br></ej-grid><br> **TS** <br>beforeBatchDelete(e: any){}| **Event:** *beforeBatchDelete* <br><br><ejs-grid (beforeBatchDelete)= 'beforeBatchDelete(\$event)'><br></ejs-grid><br> **TS** <br>beforeBatchDelete(args: any): void{}

| Triggers before saving a record in batch editing| **Event:** *beforeBatchSave* <br><br><ej-grid #grid (beforeBatchSave) = "beforeBatchSave(\$event)"><br></ej-grid><br> **TS** <br>beforeBatchSave(e: any){}| **Event:** *beforeBatchSave* <br><br><ejs-grid (beforeBatchSave)= 'beforeBatchSave(\$event)'><br></ejs-grid><br> **TS** <br>beforeBatchSave(args: any): void{}

| Triggers before the <br>record in being edited| **Event:** *beginEdit* <br><br><ej-grid #grid (beginEdit) = "beginEdit(\$event)"><br></ej-grid><br> **TS** <br>beginEdit(e: any){}| **Event:** *beginEdit* <br><br><ejs-grid (beginEdit)= 'beginEdit(\$event)'><br></ejs-grid><br> **TS** <br>beginEdit(args: any): void{}

| Triggers when the <br>cell is being edited| **Event:** *cellEdit* <br><br><ej-grid #grid (cellEdit) = "cellEdit(\$event)"><br></ej-grid><br> **TS** <br>cellEdit(e: any){}| **Event:** *cellEdit* <br><br><ejs-grid (cellEdit)= 'cellEdit(\$event)'><br></ejs-grid><br> **TS** <br>cellEdit(args: any): void{}

| Triggers when the <br>cell is saved| **Event:** *cellSave* <br><br><ej-grid #grid (cellSave) = "cellSave(\$event)"><br></ej-grid><br> **TS** <br>cellSave(e: any){}| **Event:** *cellSave* <br><br><ejs-grid (cellSave)= 'cellSave(\$event)'><br></ejs-grid><br> **TS** <br>cellSave(args: any): void{}

| Triggers when the record is added| **Event:** *endAdd* <br><br><ej-grid #grid (endAdd) = "endAdd(\$event)"><br></ej-grid><br> **TS** <br>endAdd(e: any){}| **Event:** *actionComplete* <br><br><ejs-grid (actionComplete)= 'actionComplete(\$event)'><br></ejs-grid><br> **TS** <br>actionComplete(args: any): void{}

| Triggers when the record is deleted| **Event:** *endDelete* <br><br><ej-grid #grid (endDelete) = "endDelete(\$event)"><br></ej-grid><br> **TS** <br>endDelete(e: any){}| **Event:** *actionComplete* <br><br><ejs-grid (actionComplete)= 'actionComplete(\$event)'><br></ejs-grid><br> **TS** <br>actionComplete(args: any): void{}

| Triggers when the record is edited| **Event:** *endEdit* <br><br><ej-grid #grid (endEdit) = "endEdit(\$event)"><br></ej-grid><br> **TS** <br>endEdit(e: any){}| **Event:** *actionComplete* <br><br><ejs-grid (actionComplete)= 'actionComplete(\$event)'><br></ejs-grid><br> **TS** <br>actionComplete(args: any): void{}

## Resizing

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowResizing* <br><br><ej-grid [allowResizing]="true"><br></ej-grid>|  
**Property:** *allowResizing* <br><br><ejs-grid [allowResizing]="true"><br></ejs-grid>|

| Resize a column by using the method | **Method:** *resizeColumns(column,width)* <br><br>export class  
 AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.resizeColumns("OrderID",150');<br>  
 &#160;}<br>} | **Property:** *columns.width*<br><br>To resize a column, set width to that particular column  
 and then refresh the grid by using the refresh method<br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.columns[1].width = 100;<br>this.Grid.refresh();

| Triggers when a column resize starts | **Event:** *resizeStart* <br><br><ej-grid #grid (resizeStart) =  
 "resizeStart(\$event)"><br></ej-grid><br> **TS** <br>resizeStart(e: any){} | **Event:** *resizeStart*  
 <br><br><ejs-grid (resizeStart)='resizeStart(\$event)'><br></ejs-grid><br> **TS** <br>resizeStart(args:  
 any): void{}

| Triggers when a column is resized | **Event:** *resized* <br><br><ej-grid #grid (resized) =  
 "resized(\$event)"><br></ej-grid><br> **TS** <br>resized(e: any){} | **Event:** *resizeStop* <br><br><ejs-grid  
 (resizeStop)='resizeStop(\$event)'><br></ejs-grid><br> **TS** <br>resizeStop(args: any): void{}

| Triggers when a column resize stops | **Event:** *resizeEnd* <br><br><ej-grid #grid (resizeEnd) =  
 "resizeEnd(\$event)"><br></ej-grid><br> **TS** <br>resizeEnd(e: any){} | Not Applicable

## Reordering

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowReordering* <br><br><ej-grid [allowReordering]="true"><br></ej-grid>|  
**Property:** *allowReordering* <br><br><ejs-grid [allowReordering]="true"><br></ejs-grid>|

| Reorder Columns | **Method:** *reorderColumns(fromFieldName, toFieldName)* <br><br>export class  
 AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.reorderColumns("OrderID",  
 "CustomerID");<br> &#160;}<br>} | **Method:** *reorderColumns(fromFieldName,  
 toFieldName)*<br><br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.reorderColumns("OrderID", "CustomerID");

| Reorder Rows | **Method:** *reorderRows(indexes, toIndex)* <br><br>export class AppComponent  
 {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.reorderRows([0,1],3);<br>  
 &#160;}<br>} | Not Applicable

## Context Menu

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *contextMenuSettings.enableContextMenu* <br><br><ej-grid  
[contextMenuSettings]="contextMenuSettings"></ej-grid><br> **TS**  
<br>this.contextMenuSettings = {enableContextMenu: true};| **Property:** *contextMenuItems*  
<br><br><ejs-grid [contextMenuItems]="contextMenuItems"></ejs-  
grid><br>**TS**<br>this.contextMenuItems = ['AutoFit', 'AutoFitAll'];|

| Triggers when context menu item is clicked| **Event:** *contextClick* <br><br><ej-grid #grid  
(contextClick) = "contextClick(\$event)"></ej-grid><br> **TS** <br>contextClick(e: any){}| **Event:**  
*contextMenuClick* <br><br><ejs-grid (contextMenuClick)='contextMenuClick(\$event)'></ejs-  
grid><br> **TS** <br>contextMenuClick(args: any): void{}

| Triggers when context menu opens| **Event:** *contextOpen* <br><br><ej-grid #grid (contextOpen) =  
"contextOpen(\$event)"></ej-grid><br> **TS** <br>contextOpen(e: any){}| **Event:**  
*contextMenuOpen* <br><br><ejs-grid  
(contextMenuOpen)='contextMenuOpen(\$event)'></ejs-grid><br> **TS**  
<br>contextMenuOpen(args: any): void{}

## Toolbar

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Print | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolbar: true,toolbarItems: ["print"]};| **Property:** *toolbar* <br><br><ejs-grid  
[toolbar]="toolbar"></ejs-grid><br>**TS**<br>this.toolbar = ['Print'];|

| Add | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolbar: true,toolbarItems: ["add"]};| **Property:** *toolbar* <br><br><ejs-grid  
[toolbar]="toolbar"></ejs-grid><br>**TS**<br>this.toolbar = ['Add'];|

| Edit | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolbar: true,toolbarItems: ["edit"]};| **Property:** *toolbar* <br><br><ejs-grid  
[toolbar]="toolbar"></ejs-grid><br>**TS**<br>this.toolbar = ['Edit'];|

| Delete | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolbar: true,toolbarItems: ["delete"]};| **Property:** *toolbar* <br><br><ejs-grid  
[toolbar]="toolbar"></ejs-grid><br>**TS**<br>this.toolbar = ['Delete'];|

| Update | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolbar: true,toolbarItems: ["update"]};| **Property:** *toolbar* <br><br><ejs-grid  
[toolbar]="toolbar"></ejs-grid><br>**TS**<br>this.toolbar = ['Update'];|

| Cancel | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"></ej-grid><br> **TS** <br>this.toolbarSettings =

{showToolBar: true,toolbarItems: ["cancel"]};| **Property:** *toolbar* <br><br><ej-grid  
[toolbar]="toolbar"><br></ej-grid><br>**TS**<br>this.toolbar = ['Cancel'];|

| ExcelExport | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"><br></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolBar: true,toolbarItems: ["excelExport"]};| **Property:** *toolbar* <br><br><ej-grid  
[toolbar]="toolbar"><br></ej-grid><br>**TS**<br>this.toolbar = ['ExcelExport'];|

| WordExport | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"><br></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolBar: true,toolbarItems: ["wordExport"]};| Not Applicable

| PdfExport | **Property:** *toolbarSettings.toolbarItems* <br><br><ej-grid  
[toolbarSettings]="toolbarSettings"><br></ej-grid><br> **TS** <br>this.toolbarSettings =  
{showToolBar: true,toolbarItems: ["pdfExport"]};| **Property:** *toolbar* <br><br><ej-grid  
[toolbar]="toolbar"><br></ej-grid><br>**TS**<br>this.toolbar = ['PdfExport'];|

| Refresh Toolbar | **Method:** *refreshToolBar()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshToolBar();<br> &#160;}<br>}<br>| Not Applicable

| Triggers when toolbar item is clicked | **Event:** *toolbarClick* <br><br><ej-grid #grid (toolbarClick) =  
"toolbarClick(\$event)"><br></ej-grid><br> **TS** <br>toolbarClick(e: any){}| **Event:** *toolbarClick*  
<br><br><ej-grid (toolbarClick)='toolbarClick(\$event)'"><br></ej-grid><br> **TS**  
<br>toolbarClick(args: any): void{}

## GridLines

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *gridLines* <br><br><ej-grid [gridLines]="gridLines"><br></ej-grid><br> **TS**  
<br>this.gridLines = ['Both'];| **Property:** *gridLines* <br><br><ej-grid  
[gridLines]="gridLines"><br></ej-grid><br>**TS**<br>this.gridLines = ['Both'];|

## Templates

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Detail Template | **Property:** *detailsTemplate* <br><br><ej-grid><br><ng-template  
#detailsTemplate><br>You can add template elements here<br></ng-template><br></ej-grid>|  
**Property:** *detailTemplate* <br><br><ej-grid ><br><ng-template #detailTemplate><br>You can add  
template elements here<br></ng-template><br></ej-grid>|

| Row Template | **Property:** *rowTemplate* <br><br><ej-grid><br><ng-template  
#rowTemplate><br>You can add template elements here<br></ng-template><br></ej-grid>|  
**Property:** *rowTemplate* <br><br><ej-grid ><br><ng-template #rowTemplate><br>You can add  
template elements here<br></ng-template><br></ej-grid>|

| Refresh Template | **Method:** *refreshTemplate()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshTemplate();<br> &#160;}<br>}<br>| Not Applicable

| Triggers when detail template row is clicked to collapse | **Event:** *detailsCollapse* <br><br><ej-grid #grid (detailsCollapse) = "detailsCollapse(\$event)"><br></ej-grid><br> **TS** <br>detailsCollapse(e: any){}| Not Applicable

| Triggers when detail template row is initialized | **Event:** *detailsDataBound* <br><br><ej-grid #grid (detailsDataBound) = "detailsDataBound(\$event)"><br></ej-grid><br> **TS** <br>detailsDataBound(e: any){}| Not Applicable

| Triggers when detail template<br>row is clicked to expand | **Event:** *detailsExpand* <br><br><ej-grid #grid (detailsExpand) = "detailsExpand(\$event)"><br></ej-grid><br> **TS** <br>detailsExpand(e: any){}| **Event:** *detailDataBound* <br><br><ej-grid (detailDataBound)= 'detailDataBound(\$event)'><br></ej-grid><br> **TS** <br>detailDataBound(args: any): void{}

| Triggers when refresh the template column elements in the Grid | **Event:** *templateRefresh* <br><br><ej-grid #grid (templateRefresh) = "templateRefresh(\$event)"><br></ej-grid><br> **TS** <br>templateRefresh(e: any){}| Not Applicable

### Row/Column Drag and Drop

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowRowDragAndDrop* <br><br><ej-grid [allowRowDragAndDrop]="true"><br></ej-grid> | **Property:** *allowRowDragAndDrop* <br><br><ej-grid [allowRowDragAndDrop]="true"><br></ej-grid>|

| Triggers when the row is<br>being dragged | **Event:** *rowDrag* <br><br><ej-grid #grid (rowDrag) = "rowDrag(\$event)"><br></ej-grid><br> **TS** <br>rowDrag(e: any){}| **Event:** *rowDrag* <br><br><ej-grid (rowDrag)= 'rowDrag(\$event)'><br></ej-grid><br> **TS** <br>rowDrag(args: any): void{}

| Triggers when the row drag begins | **Event:** *rowDragStart* <br><br><ej-grid #grid (rowDragStart) = "rowDragStart(\$event)"><br></ej-grid><br> **TS** <br>rowDragStart(e: any){}| **Event:** *rowDragStart* <br><br><ej-grid (rowDragStart)= 'rowDragStart(\$event)'><br></ej-grid><br> **TS** <br>rowDragStart(args: any): void{}

| Triggers when the row is dropped | **Event:** *rowDrop* <br><br><ej-grid #grid (rowDrop) = "rowDrop(\$event)"><br></ej-grid><br> **TS** <br>rowDrop(e: any){}| **Event:** *rowDrop* <br><br><ej-grid (rowDrop)= 'rowDrop(\$event)'><br></ej-grid><br> **TS** <br>rowDrop(args: any): void{}

| Triggers before the row is being dropped | **Event:** *beforeRowDrop* <br><br><ej-grid #grid (beforeRowDrop) = "beforeRowDrop(\$event)"><br></ej-grid><br> **TS** <br>beforeRowDrop(e: any){}| Not Applicable

| Triggers when the <br>column is being dragged | **Event:** *columnDrag* <br><br><ej-grid #grid (columnDrag) = "columnDrag(\$event)"><br></ej-grid><br> **TS** <br>columnDrag(e: any){}| Not Applicable

| Triggers when the <br>column drag begins | **Event:** *columnDragStart* <br><br><ej-grid #grid (columnDragStart) = "columnDragStart(\$event)"><br></ej-grid><br> **TS** <br>columnDragStart(e: any){}| Not Applicable

| Triggers when the <br>column is dropped | **Event:** *columnDrop* <br><br><ej-grid #grid (columnDrop) = "columnDrop(\$event)"><br></ej-grid><br> **TS** <br>columnDrop(e: any){}| Not Applicable

### Frozen Rows and Columns

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *scrollSettings.frozenRows* <br><br><ej-grid [allowScrolling]="true" [scrollSettings]="scrollSettings"><br></ej-grid><br> **TS** <br>this.scrollSettings = {frozenRows: 2, frozenColumn: 1}; | **Property:** *frozenRows* <br><br><ejs-grid [frozenRows]='2' [frozenColumns]='1'><br></ejs-grid>|

| isFrozen | **Property:** *columns.isFrozen* <br><br><ej-grid><br><e-columns><br><e-column field="Freight" [isFrozen]="true"><br></e-column><br></ej-grid> | **Property:** *columns.isFrozen* <br><br><ejs-grid><br><e-columns><br><e-column field='OrderID' [isFrozen]="true"><br></e-column><br></ejs-grid>|

### ForeignKey

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *columns.foreignKeyValue* <br><br><ej-grid><br><e-columns><br><e-column field="EmployeeID" foreignKeyField= "EmployeeID" foreignKeyValue= "FirstName" [dataSource]= "employeeData"><br></e-column><br></e-columns><br></ej-grid> | **Property:** *columns.foreignKeyValue* <br><br><ejs-grid><br><e-columns><br><e-column field='OrderID' foreignKeyValue='FirstName' [dataSource]='employeeData'><br></e-column><br></e-columns><br></ejs-grid>|

### Auto Wrap

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *allowTextWrap* <br><br><ej-grid [allowTextWrap]="true"><br></ej-grid> | **Property:** *allowTextWrap* <br><br><ejs-grid [allowTextWrap]='true'><br></ejs-grid>|

| Both | **Property:** *textWrapSettings.wrapMode* <br><br><ej-grid [allowTextWrap]="true" [textWrapSettings]="textWrapSettings"><br></ej-grid><br> **TS** <br>this.textWrapSettings = { wrapMode: "both"}; | **Property:** *textWrapSettings.wrapMode* <br><br><ejs-grid [allowTextWrap]='true' [textWrapSettings]='textWrapSettings'><br></ejs-grid><br> **TS** <br>this.textWrapSettings = { wrapMode: 'Both' };|



| Header | **Property:** *textWrapSettings.wrapMode* <br><br><ej-grid allowTextWrap="true" [textWrapSettings]="textWrapSettings"><br></ej-grid><br> **TS** <br>this.textWrapSettings = { wrapMode: "header"}; | **Property:** *textWrapSettings.wrapMode* <br><br><ejs-grid [allowTextWrap]='true' [textWrapSettings]='textWrapSettings'><br></ejs-grid><br>**TS**<br>this.textWrapSettings = { wrapMode: 'Header' }; |

| Content | **Property:** *textWrapSettings.wrapMode* <br><br><ej-grid [allowTextWrap]="true" [textWrapSettings]="textWrapSettings"><br></ej-grid><br> **TS** <br>this.textWrapSettings = { wrapMode: "content"}; | **Property:** *textWrapSettings.wrapMode* <br><br><ejs-grid [allowTextWrap]='true' [textWrapSettings]='textWrapSettings'><br></ejs-grid><br>**TS**<br>this.textWrapSettings = { wrapMode: 'Content' }; |

### Responsive

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *isResponsive* <br><br><ej-grid [isResPonsive]="true" [enableResponsiveRow]="true"><br></ej-grid> | Not Applicable

### State Persistence

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *enablepersistence* <br><br><ej-grid [enablepersistence]="true"><br></ej-grid> | **Property:** *enablepersistence* <br><br><ejs-grid [enablepersistence]='true' ><br></ejs-grid> |

### Right to Left - RTL

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *enableRTL* <br><br><ej-grid [enableRTL]="true" ><br></ej-grid> | **Property:** *enableRTL* <br><br><ejs-grid [enableRTL]='true' ><br></ejs-grid> |

### ToolTip

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *clipMode* <br><br><ej-grid><br><e-columns><br><e-column field="ShipName" [clipMode]="clip"><br></e-column><br><e-column field="ShipCity" [clipMode]="ellipsis"><br></e-column><br><e-column field="Freight" [clipMode]="ellipsiswithtooltip"><br></e-column><br></e-columns><br></ej-grid> | **Property:** *clipMode* <br><br><ejs-grid><br><e-columns><br><e-column field='ShipName' [clipMode]='Clip'><br></e-column><br><e-column field='ShipCity' [clipMode]='Ellipsis'><br></e-column><br><e-column field='Freight' [clipMode]='EllipsisWithTooltip'><br></e-column><br></e-columns><br></ejs-grid> |

### Aggregate/Summary

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Footer Aggregate | **Property:** *showSummary* <br><br><ej-grid [showSummary]="true"  
[summaryRows]="summaryrows"><br></ej-grid><br> **TS** <br>export class AppComponent  
{<br>public summaryrows;<br>constructor(){<br>this.summaryrows = [{<br>title:  
"Sum",<br>summaryColumns: [{<br>summaryType:  
ej.Grid.SummaryType.Sum,<br>displayColumn: "Freight",<br>dataMember:  
"Freight",<br>format: "{0:C2}"<br>}}<br>}}]} | **Property:** *aggregates* <br><br><ejs-grid><br><e-  
aggregates><br><e-aggregate><br><e-columns><br><e-column type="Sum" field="Freight"  
format="C2"><br><ng-template #footerTemplate let-data>Sum: {{data.Sum}}<br></ng-  
template><br></e-column><br></e-columns><br></e-aggregate><br></e-aggregates><br></ejs-  
grid>|

| Caption Aggregate | **Property:** *showSummary* <br><br><ej-grid [showSummary]="true"  
[summaryRows]="summaryrows"><br></ej-grid><br> **TS** <br>export class AppComponent  
{<br>public summaryrows;<br>constructor(){<br>this.summaryrows =  
[{<br>showCaptionSummary: true,<br>summaryColumns: [{<br>summaryType:  
ej.Grid.SummaryType.Average,<br>displayColumn: "Freight",<br>dataMember:  
"Freight",<br>format: "{0:C2}"<br>prefix: "Average = "<br>},<br>showTotalSummary:  
false<br>}}]} | **Property:** *aggregates* <br><br><ejs-grid><br><e-aggregates><br><e-  
aggregate><br><e-columns><br><e-column type="Sum" field="Freight" format="C2"><br><ng-  
template #groupCaptionTemplate let-data>Sum: {{data.Sum}}<br></ng-template><br></e-  
column><br></e-columns><br></e-aggregate><br></e-aggregates><br></ejs-grid>|

| Get Summary values | **Method:** *getSummaryValues(summaryCol, summaryData)* <br><br>export  
class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getSummaryValues(summaryCol,  
window.gridData);<br>&#160;}<br>} | Not Applicable

## Grid Export

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Adds a grid model property which is to be ignored on exporting<br>grid | **Method:**  
*addIgnoreOnExport(propertyNames)* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.addIgnoreOnExport(filterSettings);<br>&#160;}<br>} | Not Applicable

## Columns

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Add or Remove Columns | **Method:** *columns(columnDetails, [action])-columnDetails(array of columns  
or string of field name)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid:  
EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.columns("OrderID",  
"remove");<br>&#160;&#160;this.Grid.widget.columns("CustomerID", "add");<br>&#160;}<br>} |



**Property:** *columns* <br><br> Grid is initially rendered with OrderID and CustomerId columns. Then if you want to add ShipAddress column, you have to reset the value for column property as `this.Grid.columns = [{field:"OrderID"}, {field:"CustomerId"}, {field:"ShipAddress"}]`; Then to remove the CustomerId column, reset the column property as, `this.Grid.columns = [{field:"OrderID"}, {field:"ShipAddress"}]`;

| Get Column By Field | **Method:** `getColumnByField(fieldName)` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnByField("OrderID");<br>&#160;}<br>} | **Method:** `getColumnByField(fieldName)`<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getColumnByField("OrderID")

| Get Column By HeaderText | **Method:** `getColumnByHeaderText(headerText)` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnByHeaderText("OrderID");<br>&#160;}<br>} | You can get the column object by iterating the gridObj.columns with the corresponding headerText

| Get Column By Index | **Method:** `getColumnByIndex(columnIndex)` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnByIndex(1);<br>&#160;}<br>} | **Method:** `getColumnByIndex(columnIndex)`<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getColumnByIndex(1)

| Get Column Fieldnames | **Method:** `getColumnFieldNames()` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnFieldNames();<br>&#160;}<br>} | **Method:** `getColumnFieldNames()`<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getColumnFieldNames()

| Get Column Index By Field | **Method:** `getColumnIndexByField(fieldName)` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnIndexByField("OrderID");<br>&#160;}<br>} | **Method:** `getColumnIndexByField()`<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getColumnIndexByField("OrderID")

| Get Column Index By HeaderText | **Method:** `getColumnIndexByHeaderText(headerText, [field])`-field is optional <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getColumnIndexByHeaderText("OrderID");<br>&#160;}<br>} | You can get the Column object with the index value by iterating the gridObj.columns with headerText

| Set Width to columns | **Method:** `setWidthToColumns()` <br><br> export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.setWidthToColumns();<br>&#160;}<br>} | **Method:** `widthService.setWidthToColumns()`<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.widthService.setWidthToColumns()

| Get HeaderText By FieldName | **Method:** *getHeaderTextByFieldName()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getHeaderTextByFieldName("OrderID");<br>&#160;}<br>} | Not Applicable

| Get Hidden Column names | **Method:** *getHiddenColumnNames()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getHiddenColumnNames();<br>&#160;}<br>} | Not Applicable

| Get Visible Column name | **Method:** *getVisibleColumnNames()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getVisibleColumnNames();<br>&#160;}<br>} | **Method:** *getVisibleColumns()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getVisibleColumns()

| Select a Column | **Method:** *selectColumns(index)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.selectColumns(1);<br>&#160;}<br>} | Not Applicable

| Select the Specified Columns based on the index provided | **Method:** *selectColumns(columnIndex(fromIndex),[toIndex])* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.selectColumns(1, 4);<br>&#160;}<br>} | Not Applicable

## Row

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Enable Hover | **Property:** *enableRowHover* <br><br><ej-grid [enableRowHover]="true" ><br></ej-grid> | **Property:** *enableHover* <br><br><ejs-grid [enableHover]='true'><br></ejs-grid> |

| Get Row Height | **Method:** *getRowHeight()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getRowHeight();<br>&#160;}<br>} | **Method:** *getRowHeight()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getRowHeight();

| Refresh Row Height | **Method:** *rowHeightRefresh()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.rowHeightRefresh();<br>&#160;}<br>} | Not Applicable

| Get index by Row Element | **Method:** *getIndexByRow(\$tr)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getIndexByRow(\$(".gridcontent tr").first());<br>&#160;}<br>} | Not Applicable

| Get Row by its Index | **Method:** *getRowByIndex(from, to)* <br><br>export class AppComponent  
 {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getRowByIndex(3, 6);<br>  
 &#160;}<br>} | **Method:** *getRowByIndex(index)*<br><br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.getRowByIndex(1);

| Get rendered rows | **Method:** *getRows()* <br><br>export class AppComponent  
 {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getRows();<br> &#160;}<br>} |  
**Method:** *getRows()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getRows();

| Triggers while hover the grid row | **Event:** *rowHover* <br><br><ej-grid #grid (rowHover) =  
 "rowHover(\$event)"><br></ej-grid><br> **TS** <br>rowHover(e: any){}| Not Applicable

### Show/Hide Columns

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Hide Columns by using method | **Method:** *hideColumns(headerText)* <br><br>export class  
 AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.hideColumns("Order ID");<br>  
 &#160;}<br>} | **Method:** *hideColumns(headerText)*<br><br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.hideColumns("Order ID");

| Show Columns by using method | **Method:** *showColumns(headerText)* <br><br>export class  
 AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.showColumns("Order ID");<br>  
 &#160;}<br>} | **Method:** *showColumns(headerText)*<br><br> @ViewChild('grid') Grid:  
 GridComponent;<br>this.Grid.showColumns("Order ID");

### Column Chooser

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property:** *showColumnChooser* <br><br><ej-grid [showColumnChooser]="true"><br><e-  
 columns><br><e-column field="ShipName" [showInColumnChooser]="false"><br></e-  
 column><br><e-column field="ShipCity"><br></e-column><br></e-columns><br></ej-grid> |  
**Property:** *showColumnChooser* <br><br><ejs-grid [showColumnChooser]= 'true'><br><e-  
 columns><br><e-column field='CustomerName' [showInColumnChooser]='false'><br></e-  
 column><br><e-column field='Country'><br></e-column><br></e-columns><br></ejs-grid> |

### Header

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Refresh Header | **Method:** *refreshHeader()* <br><br>export class AppComponent  
 {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>  
 &#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshHeader();<br> &#160;}<br>} |

**Method:** *refreshHeader()* `@ViewChild('grid') Grid: GridComponent;`  
`this.Grid.refreshHeader();`

| Triggers every time a request is made to access particular header cell information, element and data. |

**Event:** *mergeHeaderCellInfo* `<ej-grid #grid (mergeHeaderCellInfo) = "mergeHeaderCellInfo($event)"></ej-grid>` **TS** `<mergeHeaderCellInfo(e: any){}` | Not Applicable

| Triggers every time a request is made to access particular cell information, element and data | **Event:** *mergeCellInfo* `<ej-grid #grid (mergeCellInfo) = mergeCellInfo($event)"></ej-grid>` **TS** `<mergeCellInfo(e: any){}` | Not Applicable

### DataSource

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| DataSource-When *templateRefresh*(optional) is set true, both header and contents get refreshed |

**Method:** *dataSource*(*newDataSource*, [*templateRefresh*]) `<export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.dataSource(newDataSource);<br>&#160;}<br>}` | **Property:** *dataSource* `@ViewChild('grid') Grid: GridComponent;`  
`this.Grid.dataSource = newDataSource;`

### Print

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Print the grid | **Method:** *print()* `<export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.print();<br>&#160;}<br>}` | **Method:** *print()* `@ViewChild('grid') Grid: GridComponent;`  
`this.Grid.print();`

| Triggers before printing the grid | **Event:** *beforePrint* `<ej-grid #grid (beforePrint) = "beforePrint($event)"></ej-grid>` **TS** `<beforePrint(e: any){}` | **Event:** *beforePrint* `<ejs-grid (beforePrint)="beforePrint($event)"></ejs-grid>` **TS** `<beforePrint(args: any): void{}`

### Scrolling

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Get ScrollObject | **Method:** *getScrollObject()* `<export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getScrollObject();<br>&#160;}<br>}` | **Property:** *grid.scrollModule* `@ViewChild('grid') Grid: GridComponent;`  
`var scrollObj = this.Grid.scrollModule;`

### PrimaryKey

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

|Set PrimaryKey Column | **Property:** *columns.isPrimaryKey* <br><br><ej-grid><br><e-columns><br><e-column field="OrderID" [isPrimaryKey]="true"><br></e-column><br></e-columns><br></ej-grid> | **Property:** *columns.isPrimaryKey* <br><br><ejs-grid><br><e-columns><br><e-column field='OrderID' [isPrimaryKey]='true'><br></e-column><br></e-columns><br></ejs-grid>|

|Get the PrimaryKey fieldnames| **Method:** *getPrimaryKeyFieldNames()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getPrimaryKeyFieldNames();<br>&#160;}<br>} | **Method:** *getPrimaryKeyFieldNames()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.getPrimaryKeyFieldNames();

## Grid

|Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

|Get the Browser Details| **Method:** *getBrowserDetails()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getBrowserDetails();<br>&#160;}<br>} | In Essential JS 2, it can be <br>achieved by using Browser class of ej2-base

|Set dimension for the grid| **Method:** *setDimension(height, width)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.setDimension(300, 400);<br>&#160;}<br>} | Not Applicable

|set maximum width for mobile| **Method:** *setPhoneModeMaxWidth(value)* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.setPhoneModeMaxWidth(500);<br>&#160;}<br>} | Not Applicable

|Render the grid| **Method:** *render()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.render();<br>&#160;}<br>} | **Method:** *render()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.render();

|Reset the model collections like pageSettings, <br>groupSettings, filterSettings, sortSettings and summaryRows| **Method:** *resetModelCollections()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.resetModelCollections();<br>&#160;}<br>} | Not Applicable

|Destroy the grid| **Method:** *destroy()* <br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.destroy();<br>&#160;}<br>} | **Method:** *destroy()*<br><br> @ViewChild('grid') Grid: GridComponent;<br>this.Grid.destroy();

| Get Content Element | **Method:** *getContent()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getContent();<br> &#160;}<br>} |  
**Method:** *getContent()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getContent();

| Get Content Table | **Method:** *getContentTable()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getContentTable();<br> &#160;}<br>} |  
**Method:** *getContentTable()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getContentTable();

| Get Current View Data | **Method:** *getCurrentViewData()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getCurrentViewData();<br>&#160;}<br>} | **Method:** *getCurrentViewRecords()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getCurrentViewRecords();

| Get Data Row | **Method:** *getDataByIndex(rowIndex)* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getDataByIndex(0);<br> &#160;}<br>} | **Method:** *getDataRows()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getDataRows();

| Get Fieldname by using the headertext | **Method:** *getFieldNameByHeaderText(headerText)*  
<br><br>export class AppComponent {<br>&#160;@ViewChild('grid') Grid: EJComponents<any,  
any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getFieldNameByHeaderText("Order  
ID");<br> &#160;}<br>} | Not Applicable

| Get FooterContent | **Method:** *getFooterContent()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getFooterContent();<br>&#160;}<br>} | **Method:** *getFooterContent()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getFooterContent();

| Get FooterContent Table | **Method:** *getFooterTable()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getFooterTable();<br> &#160;}<br>} |  
**Method:** *getFooterContentTable()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getFooterContentTable();

| Get HeaderContent | **Method:** *getHeaderContent()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getHeaderContent();<br>&#160;}<br>} | **Method:** *getHeaderContent()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getHeaderContent();

| Get HeaderContent Table | **Method:** *getHeaderTable()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.getHeaderTable();<br> &#160;}<br>} |  
**Method:** *getHeaderTable()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.getHeaderTable();

| Refresh Content | **Method:** *refreshContent()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshContent();<br> &#160;}<br>} |  
**Method:** *contentModule.refreshContentRows()*<br><br> @ViewChild('grid') Grid:  
GridComponent;<br>this.Grid.contentModule.refreshContentRows();

| Refresh Data | **Method:** *refreshData()* <br><br>export class AppComponent  
{<br>&#160;@ViewChild('grid') Grid: EJComponents<any, any>;<br>&#160;ngAfterViewInit(){<br>&#160;&#160;this.Grid.widget.refreshData();<br> &#160;}<br>} | Not  
Applicable

| Triggers every time a request is <br>made to access particular cell <br>information, element and data |  
**Event:** *queryCellInfo* <br><br><ej-grid #grid (queryCellInfo) = "queryCellInfo(\$event)"><br> TS  
<br>queryCellInfo(e: any){} | **Event:** *queryCellInfo* <br><br><ejs-grid  
(queryCellInfo)='queryCellInfo(\$event)'"><br> TS <br>queryCellInfo(args: any): void{}

| Triggers every time a request is <br>made to access row information, <br>element and data | **Event:**  
*rowDataBound* <br><br><ej-grid #grid (rowDataBound) = "rowDataBound(\$event)"><br></ej-  
grid><br> TS <br>rowDataBound(e: any){} | **Event:** *rowDataBound* <br><br><ejs-grid  
(rowDataBound)='rowDataBound(\$event)'"><br></ejs-grid><br> TS <br>rowDataBound(args:  
any): void{}

| Triggers for every grid <br>action before it get started | **Event:** *actionBegin* <br><br><ej-grid #grid  
(actionBegin) = "actionBegin(\$event)"><br></ej-grid><br> TS <br>actionBegin(e: any){} | **Event:**  
*actionBegin* <br><br><ejs-grid (actionBegin)='actionBegin(\$event)'"><br></ejs-grid><br> TS  
<br>actionBegin(args: any): void{}

| Triggers for every grid <br>action success event | **Event:** *actionComplete* <br><br><ej-grid #grid  
(actionComplete) = "actionComplete(\$event)"><br></ej-grid><br> TS <br>actionComplete(e:  
any){} | **Event:** *actionComplete* <br><br><ejs-grid  
(actionComplete)='actionComplete(\$event)'"><br></ejs-grid><br> TS <br>actionComplete(args:  
any): void{}

| Triggers for every grid <br>server failure event | **Event:** *actionFailure* <br><br><ej-grid #grid  
(actionFailure) = "actionFailure(\$event)"><br></ej-grid><br> TS <br>actionFailure(e: any){} |  
**Event:** *actionFailure* <br><br><ejs-grid (actionFailure)='actionFailure(\$event)'"><br></ejs-grid><br>  
TS <br>actionFailure(args: any): void{}

| Triggers when the grid is bound <br>with data during rendering | **Event:** *dataBound* <br><br><ej-grid  
#grid (dataBound) = "dataBound(\$event)"><br></ej-grid><br> TS <br>dataBound(e: any){} |  
**Event:** *dataBound* <br><br><ejs-grid (dataBound)='dataBound(\$event)'"><br></ejs-grid><br> TS  
<br>dataBound(args: any): void{}



| Triggers when the grid is going to destroy | **Event:** *destroy*   
 `<ej-grid #grid (destroy) = "destroy($event)"></ej-grid>` **TS** `<destroy(e: any){}` | **Event:** *destroyed*   
 `<ejs-grid (destroyed)='destroyed($event)'></ejs-grid>` **TS** `<destroyed(args: any): void{}>`

| Triggers when initial load of the grid | **Event:** *load*   
 `<ej-grid #grid (load) = "load($event)"></ej-grid>` **TS** `<load(e: any){}` | **Event:** *load*   
 `<ejs-grid (load)='load($event)'></ejs-grid>` **TS** `<load(args: any): void{}>`

| Triggers when the grid is rendered completely | **Event:** *create*   
 `<ej-grid #grid (create) = "create($event)"></ej-grid>` **TS** `<create(e: any){}` | **Event:** *created*   
 `<ejs-grid (created)='created($event)'></ejs-grid>` **TS** `<created(args: any): void{}>`

| Triggers when the record is clicked | **Event:** *recordClick*   
 `<ej-grid #grid (recordClick) = "recordClick($event)"></ej-grid>` **TS** `<recordClick(e: any){}` | Not Applicable

| Triggers when right clicked on grid element | **Event:** *rightClick*   
 `<ej-grid #grid (rightClick) = "rightClick($event)"></ej-grid>` **TS** `<rightClick(e: any){}` | Not Applicable

| Triggers when the record is double clicked | **Event:** *recordDoubleClick*   
 `<ej-grid #grid (recordDoubleClick) = "recordDoubleClick($event)"></ej-grid>` **TS** `<recordDoubleClick(e: any){}` | **Event:** *recordDoubleClick*   
 `<ejs-grid (recordDoubleClick)='recordDoubleClick($event)'></ejs-grid>` **TS** `<recordDoubleClick(args: any): void{}>`

## How To

Enable/disable grid and its actions in Angular Grid component

You can enable/disable the Grid and its actions by applying/removing corresponding CSS styles.

To enable/disable the grid and its actions, follow the given steps:

**Step 1:** Create CSS class with custom style to override the default style of Grid.

```
`css
.disablegrid {
  pointer-events: none;
  opacity: 0.4;
}
.wrapper {
  cursor: not-allowed;
}
`
```

**Step 2:** Add/Remove the CSS class to the Grid in the click event handler of Button.

```
`typescript
public btnClick():void {
  if (this.Grid.element.classList.contains('disablegrid')) {
```



```

this.Grid.element.classList.remove('disablegrid');
document.getElementById("GridParent").classList.remove('wrapper');
}
else {
this.Grid.element.classList.add('disablegrid');
document.getElementById("GridParent").classList.add('wrapper');
}
}
,

```

In the below demo, the button click will enable/disable the Grid and its actions.

#### **APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems, GridComponent } from
 '@syncfusion/ej2-angular-grids';
@Component({
imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
],
providers: [EditService, ToolbarService, SortService, PageService],
standalone: true,
    selector: 'app-root',
    template: `<button ejs-button (click)="btnClick()" cssClass="e-
flat">Enable/Disable Grid</button>
        <div id="GridParent">
            <ejs-grid #Grid [dataSource]='data'
[editSettings]='editSettings' [toolbar]='toolbar' height='273px'>
                <e-columns>
                    <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=100></e-column>

```

```

        <e-column field='CustomerID'
headerText='Customer ID' width=120></e-column>
        <e-column field='Freight' headerText='Freight'
textAlign= 'Right'
        editType= 'numericedit' width=120 format=
'C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship
Country' editType= 'dropdownedit' width=150></e-column>
    </e-columns>
</ejs-grid>
</div>`
})
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('Grid') public grid?: GridComponent;
    public editSettings?: EditSettingsModel;
    public toolbar?: ToolbarItems[];
    ngOnInit(): void {
        this.data = data;
        this.editSettings = { allowAdding: true, allowEditing: true,
allowDeleting: true };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    }
    public btnClick(): void {
        if ((this.grid as any).element.classList.contains('disablegrid')) {
            (this.grid as any).element.classList.remove('disablegrid');
            (document.getElementById('GridParent') as
any).classList.remove('wrapper');
        } else {
            (this.grid as any).element.classList.add('disablegrid');
            (document.getElementById('GridParent') as
any).classList.add('wrapper');
        }
    }
}

```

## 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 edit dialog in Angular Grid component

You can customize the appearance of the edit dialog in the [actionComplete](#) event based on **requestType** as **beginEdit** or **add**.

In the following example, the dialog's properties like header text, showCloseIcon, height have been changed while editing and adding the records.

Also the locale text for the **Save** and **Cancel** buttons has been changed by overriding the default locale strings.

You can refer the Grid [Default text](#) list for more localization.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { L10n } from '@syncfusion/ej2-base';
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { EditSettingsModel, ToolbarItems } from '@syncfusion/ej2-angular-
grids';
L10n.load({
  'en-US': {
    grid: {
      'SaveButton': 'Submit',
      'CancelButton': 'Discard'
    }
  }
});
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule
  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [editSettings]='editSettings'
[toolbar]='toolbar'
  (actionComplete)="actionComplete($event)" height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign= 'Right'
editType= 'numericedit' width=120 format= 'C2'></e-
column>
      <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' width=150></e-column>
    </e-columns>
  </ejs-grid>`

```

```

    })
    export class AppComponent implements OnInit {
        public data?: object[];
        public editSettings?: EditSettingsModel;
        public toolbar?: ToolbarItems[];
        ngOnInit(): void {
            this.data = data;
            this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Dialog' };
            this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        }
        actionComplete(args: any) {
            if ((args as any).requestType === 'beginEdit' || (args as
any).requestType === 'add')) {
                const dialog = (args as any).dialog;
                const CustomerID = 'CustomerID';
                dialog.showCloseIcon = false;
                dialog.height = 400;
                // change the header of the dialog
                dialog.header = (args as any).requestType === 'beginEdit' ?
'Edit Record of ' + (args as any).rowData['CustomerID'] : 'New Customer';
            }
        }
    }
}

```

### MAIN.TS

```

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

```

Example of angular ui grid to edit a cell using cascading drop down list in Angular Grid component

You can achieve the Cascading DropDownList with grid Editing by using the Cell Edit Template feature.

In the below demo, Cascading DropDownList rendered for **ShipCountry** and **ShipState** column.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService, PageService }
from '@syncfusion/ej2-angular-grids'
import { DatePickerAllModule } from '@syncfusion/ej2-angular-calendars'
import { TimePickerModule } from '@syncfusion/ej2-angular-calendars'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { AutoCompleteModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { Query } from '@syncfusion/ej2-data';
import { cascadeData } from './datasource';

```

```

import { EditSettingsModel, ToolbarItems, IEditCell } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    DatePickerAllModule,
    FormsModule,
    TimePickerModule,
    FormsModule,
    TextBoxModule,
    MultiSelectModule,
    AutoCompleteModule

  ],
  providers: [EditService, ToolbarService, SortService, PageService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [editSettings]='editSettings'
[toolbar]='toolbar' height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
editType= 'dropdownedit' [edit]='countryParams' width=150></e-column>
      <e-column field='ShipState' headerText='Ship State'
editType= 'dropdownedit' [edit]='stateParams' width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: Object[];
  public editSettings?: EditSettingsModel;
  public toolbar?: ToolbarItems[];
  public countryParams?: IEditCell;
  public stateParams?: IEditCell;
  public countryElem?: HTMLElement;
  public countryObj?: DropDownList;
  public stateElem?: HTMLElement;
  public stateObj?: DropDownList;
  public country: { [key: string]: Object }[] = [
    { countryName: 'United States', countryId: '1' },
    { countryName: 'Australia', countryId: '2' }
  ];
  public state: { [key: string]: Object }[] = [
    { stateName: 'New York', countryId: '1', stateId: '101' },
    { stateName: 'Virginia', countryId: '1', stateId: '102' },
    { stateName: 'Washington', countryId: '1', stateId: '103' },
    { stateName: 'Queensland', countryId: '2', stateId: '104' },
    { stateName: 'Tasmania', countryId: '2', stateId: '105' },
    { stateName: 'Victoria', countryId: '2', stateId: '106' }
  ];
  ngOnInit(): void {
    this.data = cascadeData;
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };

```

```

    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    this.countryParams = {
      create: ()=>{
        this.countryElem = document.createElement('input');
        return this.countryElem;
      },
      read: ()=>{
        return (this.countryObj as any).text;
      },
      destroy: ()=>{
        (this.countryObj as any).destroy();
      },
      write: ()=>{
        this.countryObj = new DropDownList({
          dataSource: this.country,
          fields: { value: 'countryId', text: 'countryName' },
          change: () => {
            (this.stateObj as any).enabled = true;
            let tempQuery: Query = new Query().where('countryId',
'equal', (this.countryObj as any).value);
            (this.stateObj as any).query = tempQuery;
            (this.stateObj as any).text = null;
            (this.stateObj as any).dataBind();
          },
          placeholder: 'Select a country',
          floatLabelType: 'Never'
        });
        this.countryObj.appendTo(this.countryElem);
      }
    };
    this.stateParams = {
      create: ()=>{
        this.stateElem = document.createElement('input');
        return this.stateElem;
      },
      read: ()=>{
        return (this.stateObj as any).text;
      },
      destroy: ()=>{
        (this.stateObj as any).destroy();
      },
      write: ()=>{
        this.stateObj = new DropDownList({
          dataSource: this.state,
          fields: { value: 'stateId', text: 'stateName' },
          enabled: false,
          placeholder: 'Select a state',
          floatLabelType: 'Never'
        });
        this.stateObj.appendTo(this.stateElem);
      }
    }
  }
}

```

### MAIN.TS

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

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

### Hide sorting in excel filter in Angular Grid component

You can hide the sorting options on the excel filter dialog by setting display as none for the following classes.

```
`css
.e-excel-ascending,
.e-excel-descending,
.e-separator.e-excel-separator {
display: none;
}
`
```

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    GridModule
  ],
  providers: [FilterService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[filterSettings]='filterOptions' height='273px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=100></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=100></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public filterOptions?: FilterSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.filterOptions = {
```

```

        type: 'Excel',
    };
}
}

```

## MAIN.TS

```

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

```

## Exporting grid in cordova application in Angular Grid component

Exporting the Syncfusion Angular Grid in a Cordova application can be beneficial in various scenarios where users need to generate and download reports, share data in Excel or PDF formats, or archive information for offline use. A Cordova application does not support direct file download. To export the Syncfusion Angular Grid component in a Cordova application, you need to utilize Blob streams. This can be achieved by using the appropriate exporting methods and export complete events to obtain the Blob stream.

The following example illustrates how to export a Syncfusion Angular Grid in a Cordova application. It utilizes the [excelExportComplete](#) and [pdfExportComplete](#) events to manage the export process for Excel and PDF formats and obtain the Blob stream. The `exportBlob` function is responsible for creating a downloadable link for the exported file.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ToolbarService, PdfExportService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import {
    GridComponent, ToolbarItems, ToolbarService, ExcelExportService,
    PdfExportService,
    ExcelExportCompleteArgs, PdfExportCompleteArgs
} from '@syncfusion/ej2-angular-grids';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
    imports: [

        GridModule
    ],
    providers: [PdfExportService, ToolbarService],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #grid id='Grid' [dataSource]='data'
    [toolbar]='toolbarOptions' height='272px' [allowExcelExport]='true'
    (excelExportComplete)='excelExpComplete($event)'
    (pdfExportComplete)='pdfExpComplete($event)'
    [allowPdfExport]='true' (toolbarClick)='toolbarClick($event)'>
        <e-columns>

```



```

        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
        <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
        <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
    </e-columns>
</ejs-grid>`,
    providers: [ToolbarService, ExcelExportService, PdfExportService]
})
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbarOptions?: ToolbarItems[];
    @ViewChild('grid') public grid?: GridComponent;
    public exportBlob = (blob: Blob) => {
        const a: HTMLAnchorElement = document.createElement('a');
        document.body.appendChild(a);
        a.style.display = 'none';
        const url: string = window.URL.createObjectURL(blob); // Fix typo
here
        a.href = url;
        a.download = 'Export';
        a.click();
        window.URL.revokeObjectURL(url); // Fix typo here
        document.body.removeChild(a);
    }
    ngOnInit(): void {
        this.data = data;
        this.toolbarOptions = ['PdfExport', 'ExcelExport'];
    }
    toolbarClick(args: ClickEventArgs) {
        if (args.item.id === 'Grid_pdfexport') {
            (this.grid as GridComponent).pdfExport(undefined, undefined,
undefined, true);
        }
        if (args.item.id === 'Grid_excelexport') {
            (this.grid as GridComponent).excelExport(undefined, undefined,
undefined, true);
        }
    }
    excelExpComplete(args: ExcelExportCompleteArgs) {
        // This event will be triggered when excel exporting.
        (args as any).promise.then((e: { blobData: Blob }) => {
            // In this `then` function, you can get blob data through the
arguments after promise resolved.
            this.exportBlob(e.blobData);
        });
    }
    pdfExpComplete(args: PdfExportCompleteArgs) {
        // This event will be triggered when pdf exporting.
        (args as any).promise.then((e: { blobData: Blob }) => {
            // In this `then` function, you can get blob data through the
arguments after promise resolved.
            this.exportBlob(e.blobData);
        });
    }
}

```

**MAIN.TS**

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

## Customize pager drop down in Angular Grid component

To customize default values of pager dropdown, you need to define [pageSizes](#) as array of strings.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' allowPaging='true'
  [pageSettings]='initialPage'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public initialPage?: object;
  ngOnInit(): void {
    this.data = data;
    this.initialPage = { pageSizes: ['5', '10', 'All'], };
  }
}
```

**MAIN.TS**

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

### Add params for filtering in Angular Grid component

You can customize the default settings of the components which are used in Menu filter by using params of filter property in column definition.

In the below sample, OrderID and Freight Columns are numeric columns, while open the filter dialog then you can see that NumericTextBox with spin button is displayed to change/set the filter value. Now using the params option we hide the spin button in NumericTextBox for OrderID Column.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService } from '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
import { FilterSettingsModel } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' [allowFiltering]='true'
[allowPaging]='true' [filterSettings]='filterOption'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
[filter]='filterParams' textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Name'
width=140></e-column>
      <e-column field='ShipName' headerText='ShipName'
width=140></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign='Right' format='C2' width=90></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public filterParams?: object;
  public filterOption?: FilterSettingsModel = { type: 'Menu' };
  public height = '220px';
  ngOnInit(): void {
```

```

        this.data = data;
        this.filterParams = { params: { showSpinButton: false } };
    }
}

```

### MAIN.TS

```

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

```

### Select grid rows based on certain condition in Angular Grid component

You can select the specific row in the grid based on a certain condition by using the [selectRows](#) method in the [dataBound](#) event of Grid.

In the below demo, we have selected the grid rows only when **EmployeeID** column value greater than 3.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data, columnDataType } from './datasource';
import { GridComponent, SelectionSettingsModel, RowDataBoundEventArgs } from
'@syncfusion/ej2-angular-grids';
@Component({
  imports: [

    GridModule
  ],
  providers: [PageService,
    SortService,
    FilterService,
    GroupService],
  standalone: true,
  selector: 'app-root',
  template: `<div id="GridParent">
    <ejs-grid #Grid [dataSource]="data" allowPaging='true'
[selectionSettings]='selectionOptions'
    (rowDataBound)='rowDataBound($event)'
(dataBound)='dataBound()' height='273px'>
      <e-columns>
        <e-column field='OrderID' headerText='Order ID'
textAlign='Right' isPrimaryKey='true' width=100></e-column>
        <e-column field='CustomerID'
headerText='Customer ID' width=120></e-column>
        <e-column field='EmployeeID'
headerText='Employee ID' textAlign='Right' width=120></e-column>
        <e-column field='ShipCity' headerText='Ship
City' width=120></e-column>
        <e-column field='ShipName' headerText='Ship
Name' width=150></e-column>

```

```

        </e-columns>
    </ejs-grid>
</div>`
}))
export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('Grid')
    public grid?: GridComponent;
    public selectionOptions?: SelectionSettingsModel;
    public selIndex?: number[] = [];
    ngOnInit(): void {
        this.data = data;
        this.selectionOptions = { type: 'Multiple' };
    }
    public rowDataBound(args: RowDataBoundEventArgs ): void {
        if (((args.data as columnDataType)['EmployeeID'] as number) > 3)
        {
            this.selIndex?.push(parseInt(((args.row as
Element).getAttribute('aria-rowindex') as string), 10));
        }
    }
    public dataBound(): void {
        if (this.selIndex?.length) {
            this.grid?.selectRows(this.selIndex);
            this.selIndex = [];
        }
    }
}
}

```

## MAIN.TS

```

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

```

## Get row cell index in Angular Grid component

You can get the specific row and cell index of the grid by using [rowSelected](#) event of the grid. Here, we have get the row and cell index by using [aria-rowindex](#)(get row Index from **tr** element) and [aria-colindex](#)(column index from **td** element) attribute.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, GroupService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { data } from './datasource';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
@Component({
    imports: [

        GridModule
    ],
    providers: [GroupService],

```

```

standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
(rowSelected)='rowSelected($event)' height='267px'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
      <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
      <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  @ViewChild('grid')
  public grid?: GridComponent;
  ngOnInit(): void {
    this.data = data;
  }
  rowSelected(args: any) {
    alert('row index: ' + (args as any).row.getAttribute('aria-
rowindex'));
    alert('column index: ' + (args as any).target.getAttribute('aria-
colindex'));
  }
}

```

## MAIN.TS

```

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

```

## Display null values at bottom in Angular Grid component

By default the null values are displayed at bottom of the Grid row while perform sorting in ascending order. As well as this values are displayed at top of the Grid row while perform sorting with descending order. But you can customize this default order to display the null values at always bottom row of the Grid by using [column.sortComparer](#) method.

In the below demo we have displayed the null date values at bottom of the Grid row while sorting the **OrderDate** column in both ways.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, ToolbarService, SortService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';

```

```

import { SortEventArgs } from '@syncfusion/ej2-angular-grids';
let action: string;
@Component({
  imports: [

    GridModule
  ],
  providers: [SortService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid [dataSource]='data' allowSorting='true'
(actionBegin)='actionBegin($event)' >
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
width='100'></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width='120'></e-column>
      <e-column field='OrderDate' headerText='Order Date'
format='yMd'
[sortComparer]='sortComparer' width='120'></e-
column>
      <e-column field='ShipCountry' headerText='Ship
Country' width='150'></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
  actionBegin(args: SortEventArgs) {
    if ((args as any).requestType === 'sorting') {
      action = (args as any).direction;
    }
  }
  sortComparer(reference: any, comparer: any) {
    const sortAsc = action === 'Ascending' ? true : false;
    if (sortAsc && reference === null) {
      return 1;
    } else if (sortAsc && comparer === null) {
      return -1;
    } else if (!sortAsc && reference === null) {
      return -1;
    } else if (!sortAsc && comparer === null) {
      return 1;
    } else {
      return reference - comparer;
    }
  }
}

```

### MAIN.TS

```

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

```

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

## Enable editing in single click in Angular Grid component

### Normal Editing

You can make a row editable on a single click with **Normal** mode of editing in Grid, by using the [startEdit](#) and [endEdit](#) methods.

Bind the **mouseup** event for Grid and in the event handler call the [startEdit](#) and [endEdit](#), based on the clicked target element.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, PageService, ToolbarService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { EditSettingsModel, ToolbarItems, GridComponent } from
 '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
  imports: [

    GridModule
  ],
  providers: [EditService, PageService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data'
[editSettings]='editSettings' [toolbar]='toolbar' allowPaging='true'
(load)='load()'`>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=100 isPrimaryKey='true'></e-column>
      <e-column field='CustomerID' headerText='Customer ID'
width=120></e-column>
      <e-column field='Freight' headerText='Freight'
textAlign= 'Right' width=120 format= 'C2'></e-column>
      <e-column field='ShipCountry' headerText='Ship Country'
width=150></e-column>
    </e-columns>
  </ejs-grid>`
})
export class AppComponent implements OnInit {
  public data?: object[];
  public toolbar?: ToolbarItems[];
  @ViewChild('grid')
  public grid?: GridComponent;
  public editSettings?: EditSettingsModel;
  ngOnInit(): void {
    this.data = data;
    this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Normal' };
    this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
  }
}
```



```

load() {
    (this.grid as GridComponent).element.addEventListener('mouseup', (e)
=> {
    if ((e.target as HTMLElement).classList.contains("e-rowcell")) {
    if ((this.grid as GridComponent).isEdit)
        (this.grid as GridComponent).endEdit();
        let index: number = parseInt(((e.target as
HTMLElement).getAttribute("Index") as string));
        (this.grid as GridComponent).selectRow(index);
        (this.grid as GridComponent).startEdit();
    };
    });
    }
}

```

### MAIN.TS

```

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

```

#### Open dropdown edit popup on single click

You can open the default dropdown edit popup with single click edit by focusing the dropdown element and calling the EJ2 dropdown list's [showPopup](#) method in the Grid's [actionComplete](#) event. In this demo we have used a global flag variable in the **mouseup** event to ensure if the dropdown column is the clicked edit target.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, EditService, PageService, ToolbarService } from
'@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { EditSettingsModel, ToolbarItems, GridComponent, EditEventArgs,
Column } from '@syncfusion/ej2-angular-grids';
import { data } from './datasource';
@Component({
imports: [

    GridModule
],
providers: [EditService, PageService, ToolbarService],
standalone: true,
selector: 'app-root',
template: `
    <ejs-grid #grid [dataSource]='data' [editSettings]='editSettings'
[toolbar]='toolbar' allowPaging='true' (load)='load($event)'
(actionComplete)='onActionComplete($event)'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=100 isPrimaryKey='true'></e-column>
            <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>

```

```

        <e-column field='Freight' headerText='Freight' textAlign='Right'
width=120 format='C2'></e-column>
        <e-column field='ShipCountry' headerText='Ship Country'
editType='dropdownedit' width=150></e-column>
    </e-columns>
</ejs-grid>
,
}))
export class AppComponent implements OnInit {
    public data?: object[];
    public toolbar?: ToolbarItems[];
    @ViewChild('grid')
    public grid?: GridComponent;
    public editSettings?: EditSettingsModel;
    public isDropdown = false;
    ngOnInit(): void {
        this.data = data;
        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true, mode: 'Normal' };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
    }
    load(args: Object) {
        (this.grid as GridComponent).element.addEventListener('mouseup', (e:
MouseEvent) => {
            if ((e.target as HTMLElement).classList.contains('e-rowcell')) {
                if ((this.grid as GridComponent).isEdit) {
                    (this.grid as GridComponent).endEdit();
                }
                let rowInfo = (this.grid as GridComponent).getRowInfo(e.target as
EventTarget);
                if (rowInfo && rowInfo.column && (rowInfo.column as Column).field
=== 'ShipCountry') {
                    this.isDropdown = true;
                    (this.grid as GridComponent).selectRow((rowInfo.rowIndex as
number));
                    (this.grid as GridComponent).startEdit();
                }
            }
        });
    }
    onActionComplete(args: EditEventArgs) {
        if (args.requestType === 'beginEdit' && this.isDropdown) {
            this.isDropdown = false;
            let dropdownObj = ((args.form as HTMLFormElement).querySelector('.e-
dropdownlist') as HTMLFormElement)[0];
            dropdownObj.element.focus();
            dropdownObj.showPopup();
        }
    }
}

```

## MAIN.TS

```

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

```

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

### Use custom helper inside the loop with templates in Angular Grid component

The Syncfusion Angular Grid allows you to use custom helpers inside the loop with `ng-template` directive of a column. This feature enables you to create complex templates that can incorporate additional helper functions.

To achieve this, you can use the `*ngFor` directive inside the template column to iterate through the array and the `ngClass` directive to define dynamic function.

The **Customer Rating** column includes a custom template defined using `<ng-template>`. Inside this template, `*ngFor` directive is used to iterate through the **item** array and generates `<span>` tag, displayed as stars using the CSS below:

```
`css
.e-grid .rating .star:before {
  content: '★';
}
.e-grid .rating .star {
  font-size: 132%;
  color: lightgrey;
}
`
```

The `ngClass` directive dynamically assigns classes based on the result of the **isRatingGreater** method, highlighting the star using the CSS below:

```
`css
.e-grid .rating .star.checked {
  color: #ffa600;
}
`
```

This is demonstrated in the following example.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridModule,
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `<ejs-grid #grid [dataSource]='data' [height]='300'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=90></e-column>
      <e-column field='CustomerID' headerText='Customer
ID' width=100></e-column>
      <e-column field="Rating" headerText="Customer
Rating" width="120">
        <ng-template #template let-data>
          <div class="rate">
            <div class="rating">
              <span
                *ngFor="let i of item"
                [ngClass]="{
                  checked: isRatingGreater(data.Rating,
i),
                  star: true
                }"
              ></span>
            </div>
          </div>
        </ng-template>
      </e-column>
    </e-columns>
  </ejs-grid>`
))
export class AppComponent implements OnInit {
  public data?: object[];
  public item: number[] = [1, 2, 3, 4, 5];
  isRatingGreater(dataRating: number, comparisonValue: number): boolean {
    return dataRating >= comparisonValue;
  }
  ngOnInit(): void {
    this.data = data;
  }
}

```

## MAIN.TS

```

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

```

## Resize the grid in various dimension in Angular Grid component

The Syncfusion Angular Grid component offers a friendly way to resize the grid, allowing you to adjust its width and height for improved data visualization.

To resize the grid externally, you can use an external button to modify the width of the parent element that contains the grid. This will effectively resize the grid along with its parent container.

The following example demonstrates how to resize the grid on external button click based on input.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, ResizeService } from '@syncfusion/ej2-angular-grids'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { TextBoxComponent, NumericTextBoxComponent } from '@syncfusion/ej2-
angular-inputs';
import { data } from './datasource';
@Component({
  imports: [

    GridModule,
    NumericTextBoxModule,
    ButtonModule

  ],
  providers: [ResizeService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div style="display: flex">
      <div>
        <label style="padding: 30px 17px 0 0">Enter the width: </label>
        <ejs-numerictextbox #widthTextBox min='400' max='650'
placeholder="400" step='5' width="120"></ejs-numerictextbox>
      </div>
      <div>
        <label style="padding: 30px 17px 0 0">Enter the height: </label>
        <ejs-numerictextbox #heightTextBox min='200' max='600'
placeholder="200" step='5' width="120"></ejs-numerictextbox>
      </div>
      <button
        ejs-button
        style="margin:5px 0 5px 5px"
        id="resizeButton"
        cssClass="e-outline"
        (click)="onExternalResize()"
      >
        Resize
      </button>
    </div>
    <div id="parent">
      <ejs-grid #grid style="padding: 5px 5px" [dataSource]='data '
height='100%'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
          <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
          <e-column field='ShipCountry' headerText='Ship Country'
width=100></e-column>
          <e-column field='Freight' headerText='Freight' width=80></e-column>
        </e-columns>
      </ejs-grid>
  `
})
export class AppComponent implements OnInit {
  onExternalResize() {
    // ...
  }
}

```

```

</div>`,
  })
  export class AppComponent implements OnInit {
    public data?: object[];
    @ViewChild('grid')
    public grid?: GridComponent;
    @ViewChild('widthTextBox')
    public widthTextBox?: NumericTextBoxComponent;
    @ViewChild('heightTextBox')
    public heightTextBox?: NumericTextBoxComponent;
    ngOnInit(): void {
      this.data = data;
    }
    onExternalResize() {
      const parentDiv = document.getElementById('parent');
      (parentDiv as HTMLElement).style.width = (this.widthTextBox as
NumericTextBoxComponent).element.value + 'px';
      (parentDiv as HTMLElement).style.height = (this.heightTextBox as
NumericTextBoxComponent).element.value + 'px';
    }
  }
}

```

## MAIN.TS

```

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

```

## Ngx translate pipe for header text in Angular Grid component

You can use the ngx-translate to translate the headerText of grid's column. This can be achieved by using the translate pipe for **headerText** property.

This is demonstrated in the below sample code where translate is applied to headerText property of the grid columns,

`typescript

```

import { Component } from '@angular/core';
import { TranslateService } from '@ngx-translate/core';

@Component({
  selector: 'app-container',
  template: `<ejs-grid id="Grid" [dataSource]="dataSource" allowPaging="true">
<e-columns>
<e-column field="OrderID" [isPrimaryKey]="true" headerText="{{ 'Id' | translate }}"></e-column>
<e-column field="CustomerID" headerText="{{ 'Value' | translate }}"></e-column>
</e-columns>
</ejs-grid>`

```

```

})
export class AppComponent implements OnInit {
  constructor(translate: TranslateService) {
    // Specifies the available languages to the service
    translate.addLangs(['en', 'fr'])
    // Specifies the current languages to the service
    translate.use('fr');
  }
  public dataSource: any = [
    {
      OrderID: 10248, CustomerID: 'VINET', OrderDate: new Date(8364186e5),
    },
    {
      OrderID: 10249, CustomerID: 'TOMSP', OrderDate: new Date(836505e6),
    }
  ];
}

```

Import the required modules in app.module.ts file along with translate loader function,

```

`typescript
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {GridAllModule} from '@syncfusion/ej2-angular-grids';
import {AppComponent} from './app.component';
import {TranslateLoader, TranslateModule} from '@ngx-translate/core';
import {HttpClientModule, HttpClient} from '@angular/common/http';
import {TranslateHttpLoader} from '@ngx-translate/http-loader';
export function createTranslateLoader(http: HttpClient) {
  return new TranslateHttpLoader(http, './assets/i18n/', '.json');
}
@NgModule({
  declarations: [
    AppComponent
  ],

```

```

imports: [
  HttpClientModule,
  BrowserModule,
  GridAllModule,
  TranslateModule.forRoot({
    loader: {
      provide: TranslateLoader,
      useFactory: (createTranslateLoader),
      deps: [HttpClient]
    }
  }),
],
bootstrap: [AppComponent]
})
export class AppModule { }
`

```

Then add the json file with the translation text in the required languages,

```

`json
en.json
{
  "Value": "Order ID!",
  "Id": "Customer ID"
}
`

```

```

`json
fr.json
{
  "Value": "numéro de commande!",
  "Id": "N ° de client"
}
`

```

[Angular 5 sample](#)



### Unit Jasmine testing in Angular Grid component

In Jasmine, test cases are typically executed synchronously, which means that any asynchronous code within a test case will not be executed properly. For instance, if you have a Grid component that takes time to render and populate data based on your data consumption timing, your test case might fail if it runs before the Grid has finished filling data. To prevent this situation, it is advisable to write asynchronous test cases utilizing Jasmine's Async feature.

When Async testing code relies on asynchronous operations, it's essential to ensure that the test case waits for those operations to complete before making assertions or evaluating the results. The Jasmine testing framework provides a feature called [fakeAsync](#) that addresses this. By using the `fakeAsync` function, you can simulate the passage of time and control the execution of asynchronous operations within a synchronous test. It creates a "fake" zone where time can be manipulated and controlled. To create a Jasmine test case for the Grid component, follow the below steps:

#### Step 1: Set up the Jasmine testing environment.

##### I. Check and install the node version:

You need to verify if the installed version of Node is 14 or higher. If it is below version 14, you must install a version of Node above 14. You can refer the following link to install the [node version](#). You can select the any node version is 14 or above and installed.

##### II. Create an Angular application and install the Syncfusion Grid package:

To create an Angular application and install the Syncfusion Grid package, you can refer to the [Getting started](#) documentation.

##### III. Install the Jasmine and karma:

When angular application is installed, the dependencies for Jasmine and karma should be automatically installed. To verify if the Jasmine and karma dependencies is installed, you can check the `package.json` file. If the Jasmine and karma dependencies is not installed, you can run the following command to install the Jasmine and karma dependencies using npm.

```
npm install --save-dev jasmine karma karma-jasmine karma-chrome-launcher @types/jasmine
```

##### IV. Generate the karma configuration file:

When karma dependency is installed, you can run the following command to generate the karma configuration file.

```
npx karma init
```

##### V. How to implement the Jasmine test case:

You can write the Jasmine test case in the `spec.ts` extension file. After, open the test specification file and use the `describe` function to define the test case. Within the test suite, use the `it` function to specify the individual test cases.

For more detailed information on setting up testing in Angular, refer to the official [Angular documentation](#).

### Step 2: Create a Grid component.

Use the following code to create a Grid component. You can refer to the documentation [Getting started](#) to add a Grid component in your application.

```
`typescript
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
// import your datasource file here
import { stringData } from './datasource';
@Component({
  selector: 'app-root',
  template: `<ejs-grid #Grid [dataSource]='data'>
<e-columns>
<e-column field='OrderID' headerText='Order ID' textAlign='Right' width=120></e-column>
<e-column field='CustomerID' headerText='Customer ID' width=150></e-column>
<e-column field='ShipCity' headerText='Ship City' width=150></e-column>
<e-column field='ShipName' headerText='Ship Name' width=150></e-column>
</e-columns>
</ejs-grid>`
})
export class AppComponent implements OnInit {
  @ViewChild('Grid')
  public grid: GridComponent;
  public data: object[];
  ngOnInit(): void {
    this.data = stringData;
  }
}
```

**Step 3: Write a Jasmine test case that verifies whether the Grid component successfully renders with data or not.**

#### I. How to import the testing utilities and AppComponent:

You need to import the testing utilities for the `@angular/core/testing`. After, you need to import the grid component file named is `AppComponent` for Jasmine test case file and import the grid component for `GridComponent` and `GridAllModule` from the `@syncfusion/ej2-angular-grids`.

```
`typescript
import { ComponentFixture, TestBed, fakeAsync, tick } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { GridComponent, GridAllModule } from '@syncfusion/ej2-angular-grids';
`
```

## II. Define the test suite:

The `describe` function is utilized to define the test suite. Within the `describe` function, you can use the `beforeEach` function. This function to execute the before each test case. Inside the `beforeEach` function, the `TestBed.configureTestingModule` method is used to configure the testing module. So, You need to import the `GridAllModule` in the import property and declared the `AppComponent` and `GridComponent` in the declaration property.

```
`typescript
describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [
        GridAllModule
      ],
      declarations: [
        AppComponent,
        GridComponent
      ],
    }).compileComponents();
  });
});
`
```

## III. Create an instance of AppComponent:

Using the `@ViewChild('Grid')`, you can access the instance of the `GridComponent`. Inside the `beforeEach` function, by using the `TestBed.createComponent` is create the instance of `AppComponent`.

```
`typescript
// The component variable is used to store an instance of AppComponent
```

```

let component: AppComponent;

// The fixture variable is responsible for creating and managing the testing
let fixture: ComponentFixture<AppComponent>;

beforeEach(() => {

  fixture = TestBed.createComponent(AppComponent);

  component = fixture.componentInstance;

});

```

#### IV. How to write the test case in a it block:

The `it` block is used to define a test case for the "Length of the record". It uses the `fakeAsync` function. Within the test case, the `detectChanges` method is used to trigger change detection in the component. The `tick` function is used to simulate the passage of time by updating the virtual clock by 1000 milliseconds. We check that the data grid in the data source has the appropriate number of data records. The `currentViewData` property is employed to retrieve the length of data for the current page view. By utilizing this property, one can verify the accurate population of data in the grid component.

```

`typescript

it('Length of the record', fakeAsync(async () => {

  fixture.detectChanges();

  tick(1000);

  expect(component.grid.currentViewData.length).toBe(8);

}));

```

The following example illustrates how to create the grid sample and how to writing the jasmine test case.

#### **APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule } from '@syncfusion/ej2-angular-grids'
import { PageService, SortService, FilterService, GroupService } from
 '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewChild } from '@angular/core';
import { GridComponent } from '@syncfusion/ej2-angular-grids';
import { stringData } from './datasource';
@Component({
  imports: [

    GridModule

  ],
  providers: [PageService,
    SortService,
    FilterService,

```

```

        GroupService],
standalone: true,
    selector: 'app-root',
    template: `<ejs-grid #Grid [dataSource]='data'>
        <e-columns>
            <e-column field='OrderID' headerText='Order ID'
textAlign='Right' width=120></e-column>
            <e-column field='CustomerID' headerText='Customer ID'
width=150></e-column>
            <e-column field='ShipCity' headerText='Ship City'
width=150></e-column>
            <e-column field='ShipName' headerText='Ship Name'
width=150></e-column>
        </e-columns>
    </ejs-grid>`
  })
export class AppComponent implements OnInit {
  @ViewChild('Grid')
  public grid: GridComponent;
  public data: object[];
  ngOnInit(): void {
    this.data = stringData;
  }
}

```

### APP.COMPONENT.SPEC.TS

```

import { ComponentFixture, TestBed, fakeAsync, tick } from
'@angular/core/testing';
import { AppComponent } from './app.component';
import { GridComponent, GridAllModule } from '@syncfusion/ej2-angular-
grids';
describe('AppComponent', () => {
  let component: AppComponent;
  let fixture: ComponentFixture<AppComponent>;
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [
        GridAllModule
      ],
      declarations: [
        AppComponent,
        GridComponent
      ],
    }).compileComponents();
  });
  beforeEach(() => {
    fixture = TestBed.createComponent(AppComponent);
    component = fixture.componentInstance;
  });
  it('Length of the record', fakeAsync(async () => {
    fixture.detectChanges();
    tick(1000);
    expect(component.grid.currentViewData.length).toBe(8);
  }));
});

```

**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 Jasmine test case:**

The test command can be located in the `scripts` section of the `package.json` file. It is defined in the `test`. The following command

```
,
ng test
,
```

You can find the sample of the Unit Jasmine testing in DataGrid [here](#)

*How to use angular routing*

Angular routing is a feature of the Angular framework which facilitates the control of navigation within a single-page application (SPA). It provides a mechanism for managing navigation between different components or views within an application.

In Angular routing, you can able to configure the different routes for different URL's or paths, allowing you to navigate between different sections or pages of the application. This approach is particularly advantageous for SPA's since it permits the loading of content dynamically without requiring a complete page refresh.

**1. Creating an angular application and integrating the Syncfusion Grid component:**

A simple angular project can be created by following the steps under the [getting started](#) section of this documentation by enabling the routing option.

To use Angular routing in your application, you need to create at least two components that can be navigated from one to another. You can create these components using the following command line:

```
,
ng generate component component-name
,
```

Replace **component-name** with the desired name for your component. Running this command will generate the necessary files and code for your component, including the TypeScript file, HTML template, CSS styles, and the component's test file. Repeat the above command for each component you want to create.

**2. Defining Routes:**

To define the routes create a new file `app-routing.module.ts` under the app directory. You can define the routes that correspond to different components in this file. Import the components and define an array of route objects that specifies the path and the component to be rendered for each route.

```
,
```

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';

const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
,
```

### 3. Configuring the router module:

In the root module (**app.module.ts**) file import the **AppRoutingModule** and add it to the imports array.

```
import { AboutComponent } from './about.component';
,
```

### 4. Setting up navigation:

Add links to the components and assign the anchor tag that you want to add the route to the **routerLink** attribute. Set the value of the attribute to the components to render.

Add **<router-outlet>** in the root component's template file which will be replaced with the component corresponding to the current route.

```
<nav>
<ul>
<li><a routerLink="/">Home</a></li>
<li><a routerLink="/about">Grid 1</a></li>
<li><a routerLink="/contact">Grid 2</a></li>
</ul>
</nav>
```

```
<router-outlet></router-outlet>
```

In this demonstration, Angular routing was utilized to create and define routes for multiple components. By clicking on the links labeled Grid1 and Grid2, you can easily navigate to view the respective grid components. Additionally, the Home link allows you to return to the home page.

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, FilterService, PageService, GridAllModule } from
 '@syncfusion/ej2-angular-grids'
import { MultiSelectModule, CheckBoxSelectionService, DropDownListAllModule }
 from '@syncfusion/ej2-angular-dropdowns'
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { data } from './datasource';
@Component({
  imports: [
    GridAllModule,
    MultiSelectModule,
    DropDownListAllModule,
    CheckBoxModule,
    AppRoutingModule
  ],
  providers: [FilterService, PageService, CheckBoxSelectionService],
  standalone: true,
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  public data?: object[];
  ngOnInit(): void {
    this.data = data;
  }
}
```

### **APP.COMPONENT.HTML**

```
<nav>
  <ul class="router-list">
    <li><a routerLink="/home">Home</a></li>
    <li><a routerLink="/about">Grid 1</a></li>
    <li><a routerLink="/contact">Grid 2</a></li>
  </ul>
</nav>
<router-outlet></router-outlet>
```

### **APP-ROUTING.MODULE.TS**

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
```



```
import { HomeComponent } from './home/home.component';
import { AboutComponent } from './about/about.component';
import { ContactComponent } from './contact/contact.component';
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'about', component: AboutComponent },
  { path: 'contact', component: ContactComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' }, // Optional:
  // Redirect to 'home' for the empty path
  { path: '**', redirectTo: '/home' }, // Optional: Redirect to 'home' for
  // unknown routes
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

### **ABOUT.COMPONENT.TS**

```
import { Component, OnInit } from '@angular/core';
import { data } from '../datasource';
@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {
  constructor() { }
  public data!: Object[];
  ngOnInit(): void {
    this.data = data.slice(0,5);
  }
}
```

### **ABOUT.COMPONENT.HTML**

```
<div class="control-section">
  <ejs-grid [dataSource]='data'>
    <e-columns>
      <e-column field='OrderID' headerText='Order ID' textAlign='Right'
width=90></e-column>
      <e-column field='CustomerID' headerText='Customer ID' width=120></e-
column>
      <e-column field='Freight' headerText='Freight' textAlign='Right'
format='C2' width=90></e-column>
      <e-column field='OrderDate' headerText='Order Date'
textAlign='Right' format='yMd' width=120></e-column>
    </e-columns>
  </ejs-grid>
</div>
```

### **CONTACT.COMPONENT.TS**

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

```
import { employeeData } from '../datasource';
@Component({
  selector: 'app-contact',
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.css']
})
export class ContactComponent implements OnInit {
  constructor() { }
  public data!: Object[];
  ngOnInit(): void {
    this.data = employeeData.slice(0,5);
  }
}
```

### **CONTACT.COMPONENT.HTML**

```
<div class="control-section">
  <ejs-grid [dataSource]='data' allowReordering='true'>
    <e-columns>
      <e-column field='EmployeeID' headerText='Employee ID' width='120'
textAlign='Right'></e-column>
      <e-column field='FirstName' headerText='Name' width='140'></e-
column>
      <e-column field='Title' headerText='Title' width='170'></e-column>
      <e-column field='HireDate' headerText='Hired Date' width='120'
format='yMd' textAlign='Right'></e-column>
      <e-column field='ReportsTo' headerText='Reports To' width='120'
textAlign='Right'></e-column>
    </e-columns>
  </ejs-grid>
</div>
```

### **HOME.COMPONENT.TS**

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
  constructor() { }
  ngOnInit(): void {
  }
}
```

### **HOME.COMPONENT.HTML**

```
<h2 id="homeDiv">Welcome!</h2>
```

### Customize the empty record template in Angular Grid component

The empty record template feature in the Grid allows you to use custom content such as images, text, or other components, when the grid doesn't contain any records to display. This feature replaces the default message of 'No records to display' typically shown in the grid.

To activate this feature, set the `emptyRecordTemplate` property of the Grid. The `emptyRecordTemplate` property expects the HTML element or a function that returns the HTML element.

In the following example, an image and text have been rendered as a template to indicate that the grid has no data to display.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { GridModule, PageService, EditService, ToolbarService } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { DataManager } from '@syncfusion/ej2-data';
import { data, addImage } from './datasource';
@Component({
  imports: [
    GridModule
  ],
  providers: [PageService, EditService, ToolbarService],
  standalone: true,
  selector: 'app-root',
  template: `
    <div class="control-section">
      <ejs-grid [dataSource]='data' allowPaging='true'
        [pageSettings]='pageSettings' [editSettings]='editSettings'
        [toolbar]='toolbar'>
        <e-columns>
          <e-column field='OrderID' headerText='Order ID'
            width='140' textAlign='Right' isPrimaryKey='true'
            [validationRules]='orderidrules'></e-column>
          <e-column field='CustomerID' headerText='Customer ID'
            width='140' [validationRules]='customeridrules'></e-column>
          <e-column field='Freight' headerText='Freight'
            width='140' format='C2' textAlign='Right' editType='numericedit'
            [validationRules]='freightrules'></e-column>
          <e-column field='OrderDate' headerText='Order Date'
            width='120' editType='datetimepickeredit' [format]='formatoptions'
            textAlign='Right'></e-column>
          <e-column field='ShipCountry' headerText='Ship Country'
            width='150' editType='dropdownedit' [edit]='editparams'></e-column>
        </e-columns>

        <ng-template #emptyRecordTemplate>
          <div class='emptyRecordTemplate'>
            <img [src] = "imageUrl" class="e-emptyRecord" alt="No
record">
            <span>There is no data available to display at the
moment.</span>
          </div>
        </ng-template>
      </ejs-grid>
    </div>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
    // ...
  }
}
```

```

        </div>
    </ng-template>
</ejs-grid>
</div>`,
styleUrls: ['app.css'],
encapsulation: ViewEncapsulation.None
}))
export class AppComponent implements OnInit {
    public data!: Object[];
    public imgUrl!: string;
    public pageSettings!: Object;
    public editSettings!: Object;
    public toolbar!: string[];
    public customeridrules!: Object;
    public orderidrules!: Object;
    public freightrules!: Object;
    public editparams!: Object;
    public formatoptions!: Object;
    ngOnInit(): void {
        this.data = [];
        this.imgUrl = addImage;
        this.editSettings = { allowEditing: true, allowAdding: true,
allowDeleting: true };
        this.toolbar = ['Add', 'Edit', 'Delete', 'Update', 'Cancel'];
        this.editparams = { params: { dataSource: new DataManager(data),
fields: {text: 'ShipCountry', value: 'ShipCountry'}}};
        this.pageSettings = { pageCount: 5 };
        this.customeridrules = { required: true };
        this.orderidrules = { required: true, number: true };
        this.freightrules = { required: true, number: true };
        this.formatoptions = { type: 'dateTime', format: 'M/d/y hh:mm a' };
    }
}

```

**MAIN.TS**

```

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

```

## HeatMap Chart

### Getting started with Angular Heatmap chart component

This section explains the steps required to create a heat map and demonstrates the basic usage of the heat map control.

#### Setup Angular Environment

You can use **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 Heatmap 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-heatmap](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-heatmap --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-heatmap@ngcc](#) package to the application.

```
`bash  
npm install @syncfusion/ej2-angular-heatmap@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-heatmap:"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 Heatmap Module

Import Heatmap module into Angular application(app.module.ts) from the package `@syncfusion/ej2-ng-heatmap` [src/app/app.module.ts].

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

### Add Heatmap component

Modify the template in `app.component.ts` file to render the `ej2-ng-heatmap` component [src/app/app.component.ts].

```
`javascript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'my-app',
  // specifies the template string for the Heatmap component
  template: <ejs-heatmap id='heatmap-container'></ejs-heatmap>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
```

<!-- markdownlint-disable MD033 -->

Now use the `<code>my-app</code>` in the index.html instead of default one.

```
`html
<my-app></my-app>
```

Use the `npm run start` command to run the application in the browser.

`npm start`

The below example shows a basic Heatmap.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  // specifies the template string for the HeatMap component
  template: `<ejs-heatmap id="heatmap-container"></ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
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));
```

#### Module injection

The heat map components are segregated into individual feature-wise modules. To use its feature, you need to inject its feature module using the `HeatMap.Inject()` method. In the current application, the basic heat map is modified to visualize sales revenue data for week, and the tooltip and legend features of the heat map are used. Find the relevant feature modules and descriptions as follows.

- **LegendService** - Provides the legend feature by injecting it.
- **TooltipService** - Provides the tooltip feature by injecting it.

Now, import the above-mentioned modules from the heat map package and inject them into the heat map component as follows.

``javascript`

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

```
import { AppComponent } from './app.component';
import { HeatmapComponent, Legend, Tooltip } from '@syncfusion/ej2-ng-heatmap';
@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [AppComponent, HeatMapComponent],
  bootstrap: [AppComponent],
  providers: [ HeatmapComponent, LegendService, TooltipService ]
})
`
```

### Populate heat map with data

This section explains how to populate the following two-dimensional array data to the heat map.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ];
}
```



```
}
```

**MAIN.TS**

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

**Enable axis labels**

You can add axis labels to the heat map and format those labels using the [xAxis](#) and [yAxis](#) properties.

Axis labels provide additional information about the data points populated in the heat map.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ];
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
    'Robert',
    'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
```

```
};
}
```

### MAIN.TS

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

### Add heat map title

Add a title using the [titleSettings](#) property to the heat map to provide quick information to the user about the data populated in the heat map.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [titleSettings]='titleSettings' [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal'
    }
  };
  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
```

```

        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ];
    xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert',
        'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

### MAIN.TS

```

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

```

### Enable legend

Use a legend for the heat map in the [legendSettings](#) object by setting the [visible](#) property to **true** and injecting the **Legend** module using the **HeatMap.Inject(Legend)** method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [LegendService],
    standalone: true,
    selector: 'my-app',
    template:
`<ejs-heatmap id='container' style="display:block;"
[titleSettings]='titleSettings' [dataSource]='dataSource'
[xAxis]='xAxis' [yAxis]='yAxis' [legendSettings]='legendSettings'>
</ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    titleSettings: Object = {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal'
        }
    }
};

```

```
// Data for heatmap
dataSource: Object[] = [
  [73, 39, 26, 39, 94, 0],
  [93, 58, 53, 38, 26, 68],
  [99, 28, 22, 4, 66, 90],
  [14, 26, 97, 69, 69, 3],
  [7, 46, 47, 47, 88, 6],
  [41, 55, 73, 23, 3, 79],
  [56, 69, 21, 86, 3, 33],
  [45, 7, 53, 81, 95, 79],
  [60, 77, 74, 68, 88, 51],
  [25, 25, 10, 12, 78, 14],
  [25, 56, 55, 58, 12, 82],
  [74, 33, 88, 23, 86, 59]
];
xAxis: Object = {
  labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
    'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
};
yAxis: Object = {
  labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
public legendSettings: Object = {
  visible: true,
  position: 'Right',
  showLabel: true,
  height: '150px'
};
}
```

**MAIN.TS**

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

**Add data label**

Add data labels to improve the readability of the heat map. This can be achieved by setting the [showLabel](#) property to **true** in the [cellSettings](#) object.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService],
```

```

standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[cellSettings]='cellSettings' [titleSettings]='titleSettings'
[dataSource]='dataSource'
  [xAxis]='xAxis' [yAxis]='yAxis' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal'
    }
  };

  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ];

  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert',
      'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };

  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };

  public legendSettings: Object = {
    visible: true,
    position: 'Right',
    showLabel: true,
    height: '150px'
  };

  public cellSettings: Object = {
    showLabel: true,
  };
}

```

**MAIN.TS**

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

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

### Add custom cell palette

The default palette settings of the heat map cells can be customized by using the [paletteSettings](#) property. Using the [palette](#) property in [paletteSettings](#) object, you can change the color set for the cells. You can change the color mode of the cells to [fixed](#) or [gradient](#) mode using the [type](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [titleSettings]='titleSettings'
    [dataSource]='dataSource'
    [xAxis]='xAxis' [yAxis]='yAxis' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal'
    }
  };
  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
```

```

    ];
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert',
      'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    public legendSettings: Object = {
      visible: true,
      position: 'Right',
      showLabel: true,
      height: '150px'
    };
    public cellSettings: Object = {
      showLabel: true,
    };
    public paletteSettings: Object = {
      palette: [{ value: 0, color: '#C06C84' },
        { value: 50, color: '#6C5B7B' },
        { value: 100, color: '#355C7D' },
      ]
    };
  };
}

```

### MAIN.TS

```

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

```

### Enable tooltip

The tooltip is used when you cannot display information by using the data labels due to space constraints. You can enable the tooltip by setting the [showTooltip](#) property to true and injecting the Tooltip module using the `HeatMap.Inject(Tooltip)` method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:

```

```

    `<ejs-heatmap id='container' style="display:block;" showTooltip='true'
[paletteSettings]='paletteSettings'
[cellSettings]='cellSettings' [titleSettings]='titleSettings'
[dataSource]='dataSource'
[xAxis]='xAxis' [yAxis]='yAxis' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
}
export class AppComponent{
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal'
    }
  };
  // Data for heatmap
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ];
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven', 'Michael',
'Robert',
    'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  public legendSettings: Object = {
    visible:true,
    position: 'Right',
    showLabel:true,
    height:'150px'
  };
  public cellSettings: Object = {
    showLabel: true,
  };
  public paletteSettings: Object = {
    palette: [{ value: 0, color: '#C06C84' },
    { value: 50, color: '#6C5B7B' },
    { value: 100, color: '#355C7D' },
    ]
  };
}

```



**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 Heatmap Chart](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular Heatmap Chart example](#) that shows how to render the Heatmap Chart in Angular.

**Working with data in Angular Heatmap chart component**

Heat map visualizes the JSON data and two-dimensional array data. Using the data adaptor support, data can be bound to the heat map.

**Data adaptor**

Heat map supports the following types of data binding with the adaptor support.

- Array
- Table Binding
- Cell Binding
- JSON data
- Table Binding
- Cell Binding

**Array - table binding**

This data type is a collection of one dimensional array objects, at which each inner array contains data points for an X-axis data label. This is the default data binding type for heat map. You can also directly bind the array object to the [dataSource](#) property.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    (tooltipRender)='tooltipRender($event)'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings' [showTooltip]='showTooltip'>
```

```

</ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object[] = [
      [9.5, 2.2, 4.2, 8.2, -0.5, 3.2, 5.4, 7.4, 6.2, 1.4],
      [4.3, 8.9, 10.8, 6.5, 5.1, 6.2, 7.6, 7.5, 6.1, 7.6],
      [3.9, 2.7, 2.5, 3.7, 2.6, 5.1, 5.8, 2.9, 4.5, 5.1],
      [2.4, -3.7, 4.1, 6.0, 5.0, 2.4, 3.3, 4.6, 4.3, 2.7],
      [2.0, 7.0, -4.1, 8.9, 2.7, 5.9, 5.6, 1.9, -1.7, 2.9],
      [5.4, 1.1, 6.9, 4.5, 2.9, 3.4, 1.5, -2.8, -4.6, 1.2],
      [-1.3, 3.9, 3.5, 6.6, 5.2, 7.7, 1.4, -3.6, 6.6, 4.3],
      [-1.6, 2.3, 2.9, -2.5, 1.3, 4.9, 10.1, 3.2, 4.8, 2.0],
      [10.8, -1.6, 4.0, 6.0, 7.7, 2.6, 5.6, -2.5, 3.8, -1.9],
      [6.2, 9.8, -1.5, 2.0, -1.5, 4.3, 6.7, 3.8, -1.2, 2.4],
      [1.2, 10.9, 4.0, -1.4, 2.2, 1.6, -2.6, 2.3, 1.7, 2.4],
      [5.1, -2.4, 8.2, -1.1, 3.5, 6.0, -1.3, 7.2, 9.0, 4.2]
    ];

    titleSettings: Object = {
      text: 'GDP Growth Rate for Major Economies (in Percentage)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };

    xAxis: Object = {
      labels: ['China', 'India', 'Australia', 'Mexico', 'Canada',
        'Brazil',
        'USA', 'UK', 'Germany', 'Russia', 'France', 'Japan'],
      labelRotation: 45,
      labelIntersectAction: 'None',
    };

    yAxis: Object = {
      labels: ['2008', '2009', '2010', '2011', '2012', '2013', '2014',
        '2015', '2016', '2017']
    };

    public paletteSettings: Object = {
      palette: [
        { value: -1, color: '#F0D6AD' },
        { value: 0, color: '#9da49a' },
        { value: 3.5, color: '#d7c7a7' },
        { value: 6.0, color: '#6e888f' },
        { value: 7.5, color: '#466f86' },
        { value: 10, color: '#19547B' },
      ],
    };

    public legendSettings: Object = {
      visible: false
    };

    public tooltipRender(args: ITooltipEventArgs): void {
      args.content = [args.yLabel + ' | ' + args.xLabel + ' : ' +
        args.value + ' %'];
    };

    public showTooltip: 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));
```

*Array - cell binding*

This data type is a collection of array objects that contain information about the row index, column index, and data value for each cell. You can bind the data to heat map by using the [data](#) property in the [dataSource](#) and setting the [adaptorType](#) property to `Cell`.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService ],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        labels: ['2000', '2005', '2010', '2011', '2012', '2013', '2014'],
    };
    dataSource: Object[] = [
        [0, 0, 10.75], [0, 1, 14.5], [0, 2, 25.5], [0, 3, 39.5], [0, 4,
59.75], [0, 5, 35.50], [0, 6, 75.5],
        [1, 0, 20.75], [1, 1, 35.5], [1, 2, 29.5], [1, 3, 75.5], [1, 4, 80],
[1, 5, 65], [1, 6, 85],
        [2, 0, 6], [2, 1, 18.5], [2, 2, 30.05], [2, 3, 35.5], [2, 4, 40.75],
[2, 5, 50.75], [2, 6, 65],
        [3, 0, 30.5], [3, 1, 20.5], [3, 2, 45.30], [3, 3, 50], [3, 4, 55],
[3, 5, 85.80], [3, 6, 87.5],
        [4, 0, 10.5], [4, 1, 20.75], [4, 2, 35.5], [4, 3, 35.5], [4, 4,
45.5], [4, 5, 65.], [4, 6, 75.5],
        [5, 0, 45.5], [5, 1, 20.75], [5, 2, 45.5], [5, 3, 50.75], [5, 4,
79.30], [5, 5, 84.20], [5, 6, 87.36],
        [6, 0, 26.82], [6, 1, 70], [6, 2, 75], [6, 3, 79.5], [6, 4, 88.5],
[6, 5, 89.5], [6, 6, 91.75],
        [7, 0, 15.75], [7, 1, 20.75], [7, 2, 25.5], [7, 3, 42.35], [7, 4,
45.15], [7, 5, 76.5], [7, 6, 80.5],
        [8, 0, 1.98], [8, 1, 15.23], [8, 2, 43], [8, 3, 49], [8, 4, 63.80],
[8, 5, 67.97], [8, 6, 70.52],
        [9, 0, 14.31], [9, 1, 42.87], [9, 2, 77.28], [9, 3, 77.82], [9, 4,
81.44], [9, 5, 81.92], [9, 6, 83.75],
        [10, 0, 25.5], [10, 1, 35.5], [10, 2, 40.5], [10, 3, 45.05], [10, 4,
50.5], [10, 5, 75.5], [10, 6, 90.58]
    ];
    dataSourceSettings: Object = {
        isJsonData: false,
        adaptorType: 'Cell'
    };
    public paletteSettings: Object = {
        palette: [{ color: '#3498DB' },
        { color: '#2C3E50' }
        ]
    };
    public cellSettings: Object = {
        border: {
            width: '0'
        },
        textStyle: {
            color: 'white'
        },
        format: '{value} %'
    };
    public legendSettings: Object = {
        visible: false,
    };
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = [args.yLabel + ' | ' + args.xLabel + ' : ' +
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));
```

### JSON data - table binding

In JSON table data binding, each JSON object contains an X-axis data point as row header and all the corresponding Y-axis data values. You can bind the JSON table data to the heat map using the [data](#) property in [dataSource](#). To achieve this, you should enable the [isJsonData](#) property and define the [adaptorType](#) property as `Table`. The [xDataMapping](#) property is used to map the row header in JSON data.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
    [xAxis]='xAxis' [yAxis]='yAxis'
    [cellSettings]='cellSettings' [titleSettings]='titleSettings'
    [paletteSettings]='paletteSettings'>
    </ejs-heatmap>`
})
export class AppComponent{
  titleSettings: Object = {
    text: 'Olympic Medal Achievements of most Successful Countries',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['China', 'France', 'GBR', 'Germany', 'Italy', 'Japan',
    'KOR', 'Russia', 'USA'],
    labelRotation: 45,
    labelIntersectAction: 'None',
  };
  yAxis: Object = {
    title : {text: 'Olympic Year'},
    labels: ['2000', '2004', '2008', '2012', '2016'],
  };
  dataSource: Object[] = [
```

```

    { 'Region': 'USA', '2000': 93, '2004': 101, '2008': 112, '2012': 103,
      '2016': 121 },
    { 'Region': 'GBR', '2000': 28, '2004': 30, '2008': 49, '2012': 65,
      '2016': 67 },
    { 'Region': 'China', '2000': 58, '2004': 63, '2008': 100, '2012': 91,
      '2016': 70 },
    { 'Region': 'Russia', '2000': 89, '2004': 90, '2008': 60, '2012': 69,
      '2016': 55 },
    { 'Region': 'Germany', '2000': 56, '2004': 49, '2008': 41, '2012':
      44, '2016': 42 },
    { 'Region': 'Japan', '2000': 18, '2004': 37, '2008': 25, '2012': 38,
      '2016': 41 },
    { 'Region': 'France', '2000': 38, '2004': 33, '2008': 43, '2012': 35,
      '2016': 42 },
    { 'Region': 'KOR', '2000': 28, '2004': 30, '2008': 32, '2012': 30,
      '2016': 21 },
    { 'Region': 'Italy', '2000': 34, '2004': 32, '2008': 27, '2012': 28,
      '2016': 28 }
  ];
  dataSourceSettings: Object = {
    isJsonData: true,
    adaptorType: 'Table',
    xDataMapping: 'Region',
  };
  paletteSettings: Object = {
    palette: [{ color: '#F0C27B' },
      { color: '#4B1248' }
    ],
  };
  cellSettings: Object = {
    border: {
      width: 1,
      radius: 4,
      color: 'white'
    }
  };
};
}

```

## MAIN.TS

```

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

```

### JSON data - Cell binding

In JSON cell data binding, each JSON object consists a value for each cell along with a mapping value for row and column. You can bind the JSON cell data having information for each cell to the heat map using the [data](#) property in [dataSource](#). To achieve this, you should define the [adaptorType](#) property as `Cell` and enable the [isJsonData](#) property. Now, map the fields of data by using the [valueMapping](#), [xDataMapping](#) and [yDataMapping](#) properties.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
    [xAxis]='xAxis' [yAxis]='yAxis'
    [cellSettings]='cellSettings' [titleSettings]='titleSettings'
    [paletteSettings]='paletteSettings'>
    </ejs-heatmap>`
})
export class AppComponent {
  titleSettings: Object = {
    text: 'Most Visited Destinations by International Tourist
Arrivals',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Austria', 'China', 'France', 'Germany', 'Italy',
'Mexico', 'Spain', 'Thailand', 'UK', 'USA'],
  };
  yAxis: Object = {
    labels: ['2010', '2011', '2012', '2013', '2014', '2015', '2016'],
  };
  dataSource: Object[] = [
    { 'rowid': 'France', 'columnid': '2010', 'value': '77.6' },
    { 'rowid': 'France', 'columnid': '2011', 'value': '79.4' },
    { 'rowid': 'France', 'columnid': '2012', 'value': '80.8' },
    { 'rowid': 'France', 'columnid': '2013', 'value': '86.6' },
    { 'rowid': 'France', 'columnid': '2014', 'value': '83.7' },
    { 'rowid': 'France', 'columnid': '2015', 'value': '84.5' },
    { 'rowid': 'France', 'columnid': '2016', 'value': '82.6' },
    { 'rowid': 'USA', 'columnid': '2010', 'value': '60.6' },
    { 'rowid': 'USA', 'columnid': '2011', 'value': '65.4' },
    { 'rowid': 'USA', 'columnid': '2012', 'value': '70.8' },
    { 'rowid': 'USA', 'columnid': '2013', 'value': '73.8' },
    { 'rowid': 'USA', 'columnid': '2014', 'value': '75.3' },
    { 'rowid': 'USA', 'columnid': '2015', 'value': '77.5' },
    { 'rowid': 'USA', 'columnid': '2016', 'value': '77.6' },
    { 'rowid': 'Spain', 'columnid': '2010', 'value': '64.9' },
    { 'rowid': 'Spain', 'columnid': '2011', 'value': '52.6' },
    { 'rowid': 'Spain', 'columnid': '2012', 'value': '60.8' },
    { 'rowid': 'Spain', 'columnid': '2013', 'value': '65.6' },
  ]
}

```

```

    { 'rowid': 'Spain', 'columnid': '2014', 'value': '52.6' },
    { 'rowid': 'Spain', 'columnid': '2015', 'value': '68.5' },
    { 'rowid': 'Spain', 'columnid': '2016', 'value': '75.6' },
    { 'rowid': 'China', 'columnid': '2010', 'value': '55.6' },
    { 'rowid': 'China', 'columnid': '2011', 'value': '52.3' },
    { 'rowid': 'China', 'columnid': '2012', 'value': '54.8' },
    { 'rowid': 'China', 'columnid': '2013', 'value': '51.1' },
    { 'rowid': 'China', 'columnid': '2014', 'value': '55.6' },
    { 'rowid': 'China', 'columnid': '2015', 'value': '56.9' },
    { 'rowid': 'China', 'columnid': '2016', 'value': '59.3' },
    { 'rowid': 'Italy', 'columnid': '2010', 'value': '43.6' },
    { 'rowid': 'Italy', 'columnid': '2011', 'value': '43.2' },
    { 'rowid': 'Italy', 'columnid': '2012', 'value': '55.8' },
    { 'rowid': 'Italy', 'columnid': '2013', 'value': '50.1' },
    { 'rowid': 'Italy', 'columnid': '2014', 'value': '48.5' },
    { 'rowid': 'Italy', 'columnid': '2015', 'value': '50.7' },
    { 'rowid': 'Italy', 'columnid': '2016', 'value': '52.4' },
    { 'rowid': 'UK', 'columnid': '2010', 'value': '28.2' },
    { 'rowid': 'UK', 'columnid': '2011', 'value': '31.6' },
    { 'rowid': 'UK', 'columnid': '2012', 'value': '29.8' },
    { 'rowid': 'UK', 'columnid': '2013', 'value': '33.1' },
    { 'rowid': 'UK', 'columnid': '2014', 'value': '32.6' },
    { 'rowid': 'UK', 'columnid': '2015', 'value': '34.4' },
    { 'rowid': 'UK', 'columnid': '2016', 'value': '35.8' },
    { 'rowid': 'Germany', 'columnid': '2010', 'value': '26.8' },
    { 'rowid': 'Germany', 'columnid': '2011', 'value': '29' },
    { 'rowid': 'Germany', 'columnid': '2012', 'value': '26.8' },
    { 'rowid': 'Germany', 'columnid': '2013', 'value': '27.6' },
    { 'rowid': 'Germany', 'columnid': '2014', 'value': '33' },
    { 'rowid': 'Germany', 'columnid': '2015', 'value': '35' },
    { 'rowid': 'Germany', 'columnid': '2016', 'value': '35.6' },
    { 'rowid': 'Mexico', 'columnid': '2010', 'value': '23.2' },
    { 'rowid': 'Mexico', 'columnid': '2011', 'value': '24.9' },
    { 'rowid': 'Mexico', 'columnid': '2012', 'value': '30.1' },
    { 'rowid': 'Mexico', 'columnid': '2013', 'value': '22.2' },
    { 'rowid': 'Mexico', 'columnid': '2014', 'value': '29.3' },
    { 'rowid': 'Mexico', 'columnid': '2015', 'value': '32.1' },
    { 'rowid': 'Mexico', 'columnid': '2016', 'value': '35' },
    { 'rowid': 'Thailand', 'columnid': '2010', 'value': '15.9' },
    { 'rowid': 'Thailand', 'columnid': '2011', 'value': '19.8' },
    { 'rowid': 'Thailand', 'columnid': '2012', 'value': '21.8' },
    { 'rowid': 'Thailand', 'columnid': '2013', 'value': '23.5' },
    { 'rowid': 'Thailand', 'columnid': '2014', 'value': '24.8' },
    { 'rowid': 'Thailand', 'columnid': '2015', 'value': '29.9' },
    { 'rowid': 'Thailand', 'columnid': '2016', 'value': '32.6' },
    { 'rowid': 'Austria', 'columnid': '2010', 'value': '22' },
    { 'rowid': 'Austria', 'columnid': '2011', 'value': '21.3' },
    { 'rowid': 'Austria', 'columnid': '2012', 'value': '24.2' },
    { 'rowid': 'Austria', 'columnid': '2013', 'value': '23.2' },
    { 'rowid': 'Austria', 'columnid': '2014', 'value': '25' },
    { 'rowid': 'Austria', 'columnid': '2015', 'value': '26.7' },
    { 'rowid': 'Austria', 'columnid': '2016', 'value': '28.1' },
  ];
  dataSourceSettings: Object = {
    isJsonData: true,
    adaptorType: 'Cell',
    xDataMapping: 'rowid',
  };

```



```

        yDataMapping: 'columnid',
        valueMapping: 'value'
    };
    cellSettings: Object = {
        border: {
            radius: 4,
            width: 1,
            color: 'white'
        },
        showLabel: true,
        format: '{value} M',
    };
    paletteSettings: Object = {
        palette: [{ color: '#DCD57E' },
            { color: '#A6DC7E' },
            { color: '#7EDCA2' },
            { color: '#6EB5D0' }
        ],
    };
};
}

```

### MAIN.TS

```

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

```

### Empty points

The data points that use the `null` or `""` or `undefined` as value are considered as empty points. Empty data points are ignored and not displayed in the heat map, and these points are rendered with default palette. You can customize the empty data point color value using the [emptyPointColor](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [legendSettings]='legendSettings'>
    </ejs-heatmap>`
})
export class AppComponent{

```

```

    xAxis: Object = {
      valueType: "DateTime",
      minimum: new Date(2007, 0, 1),
      intervalType: "Years",
      labelFormat: "yyyy",
    };
    yAxis: Object = {
      valueType: "Numeric"
    };
    legendSettings: Object = {
      visible: false,
    };
    dataSource: Object[] = [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, null, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, "", 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, undefined, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]
    ];
  };
}

```

### MAIN.TS

```

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

```

### Binding nested JSON data

In complex data binding, you can bind the nested JSON data to the data points in the heat map. The nested data can be mapped using the [xDataMapping](#), [yDataMapping](#), [valueMapping](#) and [bubbleDataMapping](#) properties as string value concatenated by a dot.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap';
import { LegendService, TooltipService, AdaptorService } from '@syncfusion/ej2-angular-heatmap';
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, ITooltipEventArgs } from '@syncfusion/ej2-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService ],
  standalone: true,

```

```

    selector: 'my-app',
    template:
      `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
[xAxis]='xAxis' [yAxis]='yAxis'
[cellSettings]='cellSettings' [titleSettings]='titleSettings'
[paletteSettings]='paletteSettings'>
      </ejs-heatmap>`
  })
  export class AppComponent{
    titleSettings: Object = {
      text: 'Most Visited Destinations by International Tourist
Arrivals',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Austria', 'China', 'France', 'Germany', 'Italy',
'Mexico', 'Spain', 'Thailand', 'UK', 'USA'],
    };
    yAxis: Object = {
      labels: ['2010', '2011', '2012', '2013', '2014', '2015', '2016'],
    };
    dataSource: Object[] = [
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2010' }, 'data':{ 'value':
'77.6' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2011' }, 'data':{ 'value': '79.4' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2012' }, 'data':{ 'value': '80.8' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2013' }, 'data':{ 'value': '86.6' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2014' }, 'data':{ 'value': '83.7' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2015' }, 'data':{ 'value': '84.5' }},
      { 'Labels':{ 'Xlabel': 'France', 'Ylabel': '2016' }, 'data':{ 'value': '82.6' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2010' }, 'data':{ 'value': '60.6' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2011' }, 'data':{ 'value': '65.4' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2012' }, 'data':{ 'value': '70.8' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2013' }, 'data':{ 'value': '73.8' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2014' }, 'data':{ 'value': '75.3' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2015' }, 'data':{ 'value': '77.5' }},
      { 'Labels':{ 'Xlabel': 'USA', 'Ylabel': '2016' }, 'data':{ 'value': '77.6' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2010' }, 'data':{ 'value': '64.9' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2011' }, 'data':{ 'value': '52.6' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2012' }, 'data':{ 'value': '60.8' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2013' }, 'data':{ 'value': '65.6' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2014' }, 'data':{ 'value': '52.6' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2015' }, 'data':{ 'value': '68.5' }},
      { 'Labels':{ 'Xlabel': 'Spain', 'Ylabel': '2016' }, 'data':{ 'value': '75.6' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2010' }, 'data':{ 'value': '55.6' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2011' }, 'data':{ 'value': '52.3' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2012' }, 'data':{ 'value': '54.8' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2013' }, 'data':{ 'value': '51.1' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2014' }, 'data':{ 'value': '55.6' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2015' }, 'data':{ 'value': '56.9' }},
      { 'Labels':{ 'Xlabel': 'China', 'Ylabel': '2016' }, 'data':{ 'value': '59.3' }},

```

```

{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2010'}, 'data':{ 'value': '43.6' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2011'}, 'data':{ 'value': '43.2' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2012'}, 'data':{ 'value': '55.8' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2013'}, 'data':{ 'value': '50.1' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2014'}, 'data':{ 'value': '48.5' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2015'}, 'data':{ 'value': '50.7' }},
{ 'Labels':{ 'Xlabel': 'Italy', 'Ylabel': '2016'}, 'data':{ 'value': '52.4' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2010'}, 'data':{ 'value': '28.2' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2011'}, 'data':{ 'value': '31.6' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2012'}, 'data':{ 'value': '29.8' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2013'}, 'data':{ 'value': '33.1' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2014'}, 'data':{ 'value': '32.6' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2015'}, 'data':{ 'value': '34.4' }},
{ 'Labels':{ 'Xlabel': 'UK', 'Ylabel': '2016'}, 'data':{ 'value': '35.8' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2010'}, 'data':{ 'value': '26.8' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2011'}, 'data':{ 'value': '29' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2012'}, 'data':{ 'value': '26.8' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2013'}, 'data':{ 'value': '27.6' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2014'}, 'data':{ 'value': '33' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2015'}, 'data':{ 'value': '35' }},
{ 'Labels':{ 'Xlabel': 'Germany', 'Ylabel': '2016'}, 'data':{ 'value': '35.6' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2010'}, 'data':{ 'value': '23.2' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2011'}, 'data':{ 'value': '24.9' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2012'}, 'data':{ 'value': '30.1' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2013'}, 'data':{ 'value': '22.2' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2014'}, 'data':{ 'value': '29.3' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2015'}, 'data':{ 'value': '32.1' }},
{ 'Labels':{ 'Xlabel': 'Mexico', 'Ylabel': '2016'}, 'data':{ 'value': '35' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2010'}, 'data':{ 'value': '15.9' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2011'}, 'data':{ 'value': '19.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2012'}, 'data':{ 'value': '21.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2013'}, 'data':{ 'value': '23.5' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2014'}, 'data':{ 'value': '24.8' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2015'}, 'data':{ 'value': '29.9' }},
{ 'Labels':{ 'Xlabel': 'Thailand', 'Ylabel': '2016'}, 'data':{ 'value': '32.6' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2010'}, 'data':{ 'value': '22' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2011'}, 'data':{ 'value': '21.3' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2012'}, 'data':{ 'value': '24.2' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2013'}, 'data':{ 'value': '23.2' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2014'}, 'data':{ 'value': '25' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2015'}, 'data':{ 'value': '26.7' }},
{ 'Labels':{ 'Xlabel': 'Austria', 'Ylabel': '2016'}, 'data':{ 'value': '28.1' }}
];

dataSourceSettings: Object = {
  isJsonData: true,
  adaptorType: 'Cell',
  xDataMapping: 'Labels.Xlabel',
  yDataMapping: 'Labels.Ylabel',
  valueMapping: 'data.value'
};

cellSettings: Object = {
  border: {
    radius: 4,
    width: 1,
    color: 'white'
  },
  showLabel: true,

```

```

        format: '{value} M',
    };
    paletteSettings: Object = {
        palette: [{ color: '#DCD57E' },
        { color: '#A6DC7E' },
        { color: '#7EDCA2' },
        { color: '#6EB5D0' }
        ],
    };
}

```

## 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

- [To bind data for bubble heat map with size and color attributes](#)

## Bubble heatmap in Angular Heatmap chart component

Data points represent the data source values with **gradient** or **fixed** colors in the HeatMap. You can customize the appearance of these data points by changing the **color** and **size** attributes.

The data points can be represented in color fill or bubble shape by defining the **tileType** property. By default, the data points are color filled with gradient or fixed colors and this depiction of data points is defined as **Rect** in the **tileType** property.

The cell customizations and color mapping for rect tile type is defined in [appearance](#) and [palette](#) sections in detail.

### Bubble types

The data points can be represented in the bubble along with its attributes by setting the **tileType** property to **Bubble**.

In bubble HeatMap, you can display the data points with bubble size, bubble colors, and sector attributes of the bubble.

### Bubble size

In this bubble HeatMap type, the size factor of the bubble is used to denote the data variations. The radius of the bubble varies according to data values.

By default, the bubble with small size denotes the data value with small magnitude and the larger bubble size denotes the data value with larger magnitude. This behavior can be inverted by using the [isinversedbubblesize](#) property.

To render a bubble HeatMap with size series, set the **bubbleType** property to **Size**.

## APP.COMPONENT.TS

```

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

```

```

import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

    public cellSettings: Object = {
      border: {
        width: 1
      },
      tileType: 'Bubble',
      bubbleType: 'Size',
      showLabel: false
    };
    public legendSettings: Object = {
      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));

```

### Bubble color

In HeatMap, defined with this tile type, the data points will be represented with same sized bubbles and the data value variations are represented with the bubble colors.

To represent the data points with variations in bubble colors, set the [bubbleType](#) property to **Color**.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],

```

```

[41, 55, 73, 23, 3, 79],
[56, 69, 21, 86, 3, 33],
[45, 7, 53, 81, 95, 79],
[60, 77, 74, 68, 88, 51],
[25, 25, 10, 12, 78, 14],
[25, 56, 55, 58, 12, 82],
[74, 33, 88, 23, 86, 59]];
titleSettings: Object = {
  text: 'Sales Revenue per Employee (in 1000 US$)',
  textStyle: {
    size: '15px',
    fontWeight: '500',
    fontStyle: 'Normal',
    fontFamily: 'Segoe UI'
  }
};
xAxis: Object = {
  labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
  labelRotation: 45,
  labelIntersectAction: 'None'
};
yAxis: Object = {
  labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
public paletteSettings: Object = {
  palette: [
    { color: '#C06C84'},
    { color: '#6C5B7B'},
    { color: '#355C7D'}
  ],
  type: "Gradient"
};
public cellSettings: Object = {
  border: {
    width: 1
  },
  tileType: 'Bubble',
  bubbleType: 'Color'
};
public legendSettings: Object = {
  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));

```



*Bubble sector*

In this bubble HeatMap type, the sector of the bubble decides the magnitude of data point. If the sector is large, then the data point value will be high.

To render the data points with bubble sector, set the [bubbleType](#) property to **Sector**.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
}
```

```

    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    public paletteSettings: Object = {
      palette: [
        { color: '#C06C84'},
        { color: '#6C5B7B'},
        { color: '#355C7D'}
      ],
      type: "Gradient"
    };
    public cellSettings: Object = {
      border: {
        width: 1
      },
      tileType: 'Bubble',
      bubbleType: 'Sector'
    };
    public legendSettings: Object = {
      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));

```

### Combining size and color bubble types

In this bubble HeatMap type, size and color of the bubble represents the data value variation. To render this bubble HeatMap type, set the [bubbleType](#) property to **SizeAndColor**.

The following examples demonstrate different data binding with the **SizeAndColor** bubble type set in the HeatMap.

```
<!-- markdownlint-disable MD036 -->
```

#### Array binding

When an array of numbers is specified as the data source, the bubble HeatMap can be rendered with different sizes and colors depending on the bound data.

```
<!-- markdownlint-disable MD036 -->
```

#### Table

The following example illustrates how to render a bubble HeatMap with different sizes and colors using array table binding.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap';

```

```

import { LegendService, AdaptorService, TooltipService } from
'@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { BubbleTooltipData, ITooltipEventArgs } from '@syncfusion/ej2-
angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
[cellSettings]='cellSettings' [legendSettings]='legendSettings'
(tooltipRender)='tooltipRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [[4, 39], [3, 8], [1, 3], [1, 10], [4, 4], [2, 15]],
    [[4, 28], [5, 92], [5, 73], [3, 1], [3, 4], [4, 126]],
    [[4, 45], [5, 152], [0, 44], [4, 54], [5, 243], [2, 45]]
  ];
  titleSettings: Object = {
    text: 'Commercial Aviation Accidents and Fatalities by year 2012 -
2017',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['2017', '2016', '2015'],
  };
  yAxis: Object = {
    labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec'],
  };
  public paletteSettings: Object = {
    palette: [
      { color: '#C06C84' },
      { color: '#6C5B7B' },
      { color: '#355C7D' }
    ],
    type: "Gradient"
  };
  public cellSettings: Object = {
    border: {
      width: 1
    },
    tileType: 'Bubble',

```

```

        bubbleType: 'SizeAndColor'
    };
    public legendSettings: Object = {
        visible: true,
    };
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' + 'Months ' +
            ' : ' + args.yLabel + '<br/>'
            + 'Accidents ' + ' : ' + (args.value as
BubbleTooltipData[])[0].bubbleData + '<br/>' + 'Fatalities ' + ' : '
            + (args.value as BubbleTooltipData[])[1].bubbleData];
    };
}

```

**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 -->

**Cell**

The following example illustrates how to render a bubble HeatMap with different sizes and colors using array cell binding.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { BubbleTooltipData, ITooltipEventArgs } from '@syncfusion/ej2-
angular-heatmap';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [ LegendService, AdaptorService, TooltipService],
    standalone: true,
    selector: 'my-app',
    template:
        `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
[xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
[cellSettings]='cellSettings' [legendSettings]='legendSettings'
(tooltipRender)='tooltipRender($event)'>
        </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {

```

```

    dataSource: Object[] = [
        [0, 0, [4, 39]], [0, 1, [3, 8]], [0, 2, [1, 3]], [0, 3, [1, 10]], [0,
4, [4, 4]], [0, 5, [2, 15]],
        [1, 0, [4, 28]], [1, 1, [5, 92]], [1, 2, [5, 73]], [1, 3, [3, 1]],
[1, 4, [3, 4]], [1, 5, [4, 126]],
        [2, 0, [4, 45]], [2, 1, [5, 152]], [2, 2, [0, 44]], [2, 3, [4, 54]],
[2, 4, [5, 243]], [2, 5, [2, 45]]
    ];
    dataSourceSettings: Object = {
        isJsonData: false,
        adaptorType: 'Cell'
    };
    titleSettings: Object = {
        text: 'Commercial Aviation Accidents and Fatalities by year 2012 -
2017',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['2017', '2016', '2015'],
    };
    yAxis: Object = {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec'],
    };
    public paletteSettings: Object = {
        palette: [
            { color: '#C06C84' },
            { color: '#6C5B7B' },
            { color: '#355C7D' }
        ],
        type: "Gradient"
    };
    public cellSettings: Object = {
        border: {
            width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'SizeAndColor'
    };
    public legendSettings: Object = {
        visible: true,
    };
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' + 'Months ' +
' : ' + args.yLabel + '<br/>'
            + 'Accidents ' + ' : ' + (args.value as
BubbleTooltipData[])[0].bubbleData + '<br/>' + 'Fatalities ' + ' : '
            + (args.value as BubbleTooltipData[])[1].bubbleData];
    };
}

```

**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 -->
```

**JSON binding**

When a list of JSON objects are specified as data source, the bubble HeatMap can be rendered with different sizes and colors depending on the bound data.

```
<!-- markdownlint-disable MD036 -->
```

**Table**

The following example illustrates how to render a bubble HeatMap with different sizes and colors using JSON table binding.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { BubbleTooltipData, ITooltipEventArgs } from '@syncfusion/ej2-
angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
    [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [legendSettings]='legendSettings'
    (tooltipRender)='tooltipRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    { 'Year': '2017', 'Jan-Feb': [4, 39], 'Mar-Apr': [3, 8], 'May-Jun':
    [1, 3], 'Jul-Aug': [1, 10], 'Sep-Oct': [4, 4], 'Nov-Dec': [2, 15] },
    { 'Year': '2016', 'Jan-Feb': [4, 28], 'Mar-Apr': [5, 92], 'May-Jun':
    [5, 73], 'Jul-Aug': [3, 1], 'Sep-Oct': [3, 4], 'Nov-Dec': [4, 126] },
    { 'Year': '2015', 'Jan-Feb': [4, 45], 'Mar-Apr': [5, 152], 'May-Jun':
    [0, 44], 'Jul-Aug': [4, 54], 'Sep-Oct': [5, 243], 'Nov-Dec': [2, 45] }
  ];
  dataSourceSettings: Object = {
```

```

        isJsonData: true,
        adaptorType: 'Table',
        xDataMapping: 'Year',
    };
    titleSettings: Object = {
        text: 'Commercial Aviation Accidents and Fatalities by year 2012 -
2017',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['2017', '2016', '2015'],
    };
    yAxis: Object = {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec'],
    };
    public paletteSettings: Object = {
        palette: [
            { color: '#C06C84' },
            { color: '#6C5B7B' },
            { color: '#355C7D' }
        ],
        type: "Gradient"
    };
    public cellSettings: Object = {
        border: {
            width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'SizeAndColor'
    };
    public legendSettings: Object = {
        visible: true,
    };
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' + 'Months ' +
' : ' + args.yLabel + '<br/>'
            + 'Accidents ' + ' : ' + (args.value as
BubbleTooltipData[])[0].bubbleData + '<br/>' + 'Fatalities ' + ' : '
            + (args.value as BubbleTooltipData[])[1].bubbleData];
    };
}

```

## 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 -->

## Cell

The following example illustrates how to render a bubble HeatMap with different sizes and colors using JSON cell binding.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { BubbleTooltipData, ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
    [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [legendSettings]='legendSettings'
    (tooltipRender)='tooltipRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    { Year: '2017', Months: 'Jan-Feb', Accidents: 4, Fatalities: 39 },
    { Year: '2017', Months: 'Mar-Apr', Accidents: 3, Fatalities: 8 },
    { Year: '2017', Months: 'May-Jun', Accidents: 1, Fatalities: 3 },
    { Year: '2017', Months: 'Jul-Aug', Accidents: 1, Fatalities: 10 },
    { Year: '2017', Months: 'Sep-Oct', Accidents: 4, Fatalities: 4 },
    { Year: '2017', Months: 'Nov-Dec', Accidents: 2, Fatalities: 15 },
    { Year: '2016', Months: 'Jan-Feb', Accidents: 4, Fatalities: 28 },
    { Year: '2016', Months: 'Mar-Apr', Accidents: 5, Fatalities: 92 },
    { Year: '2016', Months: 'May-Jun', Accidents: 5, Fatalities: 73 },
    { Year: '2016', Months: 'Jul-Aug', Accidents: 3, Fatalities: 1 },
    { Year: '2016', Months: 'Sep-Oct', Accidents: 3, Fatalities: 4 },
    { Year: '2016', Months: 'Nov-Dec', Accidents: 4, Fatalities: 126 },
    { Year: '2015', Months: 'Jan-Feb', Accidents: 4, Fatalities: 45 },
    { Year: '2015', Months: 'Mar-Apr', Accidents: 5, Fatalities: 152 },
    { Year: '2015', Months: 'May-Jun', Accidents: 0, Fatalities: 0 },
    { Year: '2015', Months: 'Jul-Aug', Accidents: 4, Fatalities: 54 },
    { Year: '2015', Months: 'Sep-Oct', Accidents: 5, Fatalities: 243 },
    { Year: '2015', Months: 'Nov-Dec', Accidents: 2, Fatalities: 45 },
  ];
  dataSourceSettings: Object = {
    isJsonData: true,
    adaptorType: 'Cell',
  }
}
```



```

        xDataMapping: 'Year',
        yDataMapping: 'Months',
        bubbleDataMapping: { size: 'Accidents', color: 'Fatalities' }
    };
    titleSettings: Object = {
        text: 'Commercial Aviation Accidents and Fatalities by year 2012 -
2017',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['2017', '2016', '2015'],
    };
    yAxis: Object = {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec'],
    };
    public paletteSettings: Object = {
        palette: [
            { color: '#C06C84' },
            { color: '#6C5B7B' },
            { color: '#355C7D' }
        ],
        type: "Gradient"
    };
    public cellSettings: Object = {
        border: {
            width: 1
        },
        tileType: 'Bubble',
        bubbleType: 'SizeAndColor'
    };
    public legendSettings: Object = {
        visible: true,
    };
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' + 'Months ' +
' : ' + args.yLabel + '<br/>'
            + 'Accidents ' + ' : ' + (args.value as
BubbleTooltipData[])[0].bubbleData + '<br/>' + 'Fatalities ' + ' : '
            + (args.value as BubbleTooltipData[])[1].bubbleData];
    };
}

```

## 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 -->
```

### Binding size and color values from datasource

The size and color of the bubbles in the **SizeAndColor** bubble HeatMap type can be customized by binding the datasource field name that holds the size and color values to the [size](#) and [color](#) properties in the [bubbleDataMapping](#).

The `bubbleDataMapping` supports only for the JSON data with cell adaptor type.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, AdaptorService, TooltipService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { BubbleTooltipData, ITooltipEventArgs } from '@syncfusion/ej2-
angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, AdaptorService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [dataSourceSettings]='dataSourceSettings'
    [xAxis]='xAxis' [yAxis]='yAxis'
      [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [cellSettings]='cellSettings' [legendSettings]='legendSettings'
    (tooltipRender)= 'tooltipRender($event)'`>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    { Year: '2017', Months: 'Jan-Feb', Accidents: 4, Fatalities: 39 },
    { Year: '2017', Months: 'Mar-Apr', Accidents: 3, Fatalities: 8 },
    { Year: '2017', Months: 'May-Jun', Accidents: 1, Fatalities: 3 },
    { Year: '2017', Months: 'Jul-Aug', Accidents: 1, Fatalities: 10 },
    { Year: '2017', Months: 'Sep-Oct', Accidents: 4, Fatalities: 4 },
    { Year: '2017', Months: 'Nov-Dec', Accidents: 2, Fatalities: 15 },
    { Year: '2016', Months: 'Jan-Feb', Accidents: 4, Fatalities: 28 },
    { Year: '2016', Months: 'Mar-Apr', Accidents: 5, Fatalities: 92 },
    { Year: '2016', Months: 'May-Jun', Accidents: 5, Fatalities: 73 },
    { Year: '2016', Months: 'Jul-Aug', Accidents: 3, Fatalities: 1 },
    { Year: '2016', Months: 'Sep-Oct', Accidents: 3, Fatalities: 4 },
    { Year: '2016', Months: 'Nov-Dec', Accidents: 4, Fatalities: 126 },
    { Year: '2015', Months: 'Jan-Feb', Accidents: 4, Fatalities: 45 },
    { Year: '2015', Months: 'Mar-Apr', Accidents: 5, Fatalities: 152 },
    { Year: '2015', Months: 'May-Jun', Accidents: 0, Fatalities: 0 },
    { Year: '2015', Months: 'Jul-Aug', Accidents: 4, Fatalities: 54 },
    { Year: '2015', Months: 'Sep-Oct', Accidents: 5, Fatalities: 243 },
    { Year: '2015', Months: 'Nov-Dec', Accidents: 2, Fatalities: 45 },
```

```

];
dataSourceSettings: Object = {
  isJsonData: true,
  adaptorType: 'Cell',
  xDataMapping: 'Year',
  yDataMapping: 'Months',
  bubbleDataMapping: { size: 'Accidents', color: 'Fatalities' }
};
titleSettings: Object = {
  text: 'Commercial Aviation Accidents and Fatalities by year 2012 -
2017',
  textStyle: {
    size: '15px',
    fontWeight: '500',
    fontStyle: 'Normal',
    fontFamily: 'Segoe UI'
  }
};
xAxis: Object = {
  labels: ['2017', '2016', '2015'],
};
yAxis: Object = {
  labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec'],
};
public paletteSettings: Object = {
  palette: [
    { color: '#C06C84' },
    { color: '#6C5B7B' },
    { color: '#355C7D' }
  ],
  type: "Gradient"
};
public cellSettings: Object = {
  border: {
    width: 1
  },
  tileType: 'Bubble',
  bubbleType: 'SizeAndColor'
};
public legendSettings: Object = {
  visible: true,
};
public tooltipRender(args: ITooltipEventArgs): void {
  args.content = ['Year ' + ' : ' + args.xLabel + '<br/>' + 'Months ' +
' : ' + args.yLabel + '<br/>'
    + 'Accidents ' + ' : ' + (args.value as
BubbleTooltipData[])[0].bubbleData + '<br/>' + 'Fatalities ' + ' : '
    + (args.value as BubbleTooltipData[])[1].bubbleData];
};
}

```

**MAIN.TS**

```

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

```

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

### Rendering mode in Angular Heatmap chart component

Heat map can be displayed using **Canvas** or **Scalable Vector Graphics (SVG)** rendering logic to improve the initial load performance and scalability. Heat map can also be automatically switched between **Canvas** and **SVG** modes based on dataset size. You can enable this mode by setting the [renderingMode](#) property to **Auto**.

If the **Auto** mode is enabled in the heat map and there are more than 10,000 data points, then the heat map will be rendered in a **Canvas** mode; Otherwise, the heat map will be rendered in a **SVG** mode.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [renderingMode]='renderingMode' [titleSettings]='titleSettings'
    [cellSettings]='cellSettings' [showTooltip]='showTooltip'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]
  ];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
```

```

        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    };
    public cellSettings: Object = {
        showLabel: true,
    };
    xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
            'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    public renderingMode: String = "SVG";
    public showTooltip: 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));

```

### Axis in Angular HeatMap chart component

HeatMap consists of two axes namely, X-axis and Y-axis that displays the row headers and column headers to plot the data points respectively. You can define the type, format, and other customizing options for both axes in the HeatMap.

#### Types

There are three different axis types available in the HeatMap, which defines the data type of the axis labels. You can define the axis type by using the [valueType](#) property in the HeatMap.

#### Category axis

Category axis type is used to represent the string values in axis labels.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
    '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [ LegendService, TooltipService, AdaptorService ],
    standalone: true,
    selector: 'my-app',
    template:

```

```

    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'>
    </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object [] = [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]];
    xAxis: Object = {
      valueType:"Category",
      labels: ['Nancy', 'Andrew','Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    yAxis: Object = {
      valueType:"Category",
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
  }

```

## MAIN.TS

```

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

```

### Numeric axis

Numeric axis type is used to represent the numeric values in axis labels.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService],
  standalone: true,
  selector: 'my-app',
  template:

```

```

    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'>
    </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object [] = [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]];
    xAxis: Object = {
      valueType:"Numeric",
      minimum: 0,
      maximum: 11,
    };
    yAxis: Object = {
      valueType:"Numeric",
      minimum: 0,
      maximum: 5,
    };
  }

```

## 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 axis

Date-time axis type is used to represent the date-time values in axis labels with a specific format. In date-time axis, you can define the start and end date/time using the [minimum](#) and [maximum](#) properties.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
'@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],

```

```

providers: [ LegendService, TooltipService, AdaptorService],
standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' [legendSettings]='legendSettings'
[cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
dataSource: Object[] = [
    [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
30529, 33298, 36985],
    [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
34196, 35302, 35703],
    [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
32645, 31539, 32981],
    [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
33437, 30659, 31965],
    [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
34094, 32256, 33699],
    [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
30624, 32398, 33522],
    [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
34115, 31072, 33939],
    [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
35277, 31281, 35411],
    [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
37443, 32457, 37304],
    [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
36505, 29576, 36450],
    [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
36583, 32408, 37108]
];
  titleSettings: Object = {
    text: 'Monthly Flight Traffic at JFK Airport',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    valueType: "DateTime",
    minimum: new Date(2007,0,1),
    maximum: new Date(2017,0,1),
    intervalType: "Years",
    labelFormat: "yyyy",
    labelRotation: 45,
    labelIntersectAction: 'None'
  };
  yAxis: Object = {
    labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'],

```



```

    };
    public legendSettings: Object = {
        visible: false,
    };
    public cellSettings: Object = {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    };
}

```

### MAIN.TS

```

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

```

### Inversed axis

HeatMap supports inverting the axis origin for both axes, where the axis labels are placed in an inversed manner. You can enable axis inverting by enabling the [isInversed](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [ LegendService, TooltipService, AdaptorService ],
    standalone: true,
    selector: 'my-app',
    template:
        `<ejs-heatmap id='container' style="display:block;"
        [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
        [titleSettings]='titleSettings' [legendSettings]='legendSettings'
        [cellSettings]='cellSettings'>
        </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    dataSource: Object [] = [
        [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
        30529, 33298, 36985],
        [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
        34196, 35302, 35703],
        [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
        32645, 31539, 32981],
    ]
}

```

```

        [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
        33437, 30659, 31965],
        [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
        34094, 32256, 33699],
        [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
        30624, 32398, 33522],
        [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
        34115, 31072, 33939],
        [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
        35277, 31281, 35411],
        [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
        37443, 32457, 37304],
        [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
        36505, 29576, 36450],
        [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
        36583, 32408, 37108]
    ];
    titleSettings: Object = {
        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
        labelRotation: 45,
        labelIntersectAction: 'None',
        isInversed: true
    };
    yAxis: Object = {
        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
            'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'],
    };
    public legendSettings: Object = {
        visible: false,
    };
    public cellSettings: Object = {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    };
}

```

**MAIN.TS**

```

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

```

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

### Opposed axis

In HeatMap, you can place the axis label in an opposite position of its default axis label position by using the [opposedPosition](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService, AdaptorService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [legendSettings]='legendSettings'
    [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object [] = [
    [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
    30529, 33298, 36985],
    [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
    34196, 35302, 35703],
    [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
    32645, 31539, 32981],
    [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
    33437, 30659, 31965],
    [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
    34094, 32256, 33699],
    [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
    30624, 32398, 33522],
    [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
    34115, 31072, 33939],
    [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
    35277, 31281, 35411],
    [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
    37443, 32457, 37304],
    [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
    36505, 29576, 36450],
    [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
    36583, 32408, 37108]
  ];
  titleSettings: Object = {
```

```

        text: 'Monthly Flight Traffic at JFK Airport',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        maximum: new Date(2017, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
        labelRotation: 45,
        labelIntersectAction: 'None',
        opposedPosition: true
    };
    yAxis: Object = {
        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
            'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'],
    };
    public legendSettings: Object = {
        visible: false,
    };
    public cellSettings: Object = {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    };
}

```

### MAIN.TS

```

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

```

### Axis labels customization

#### Customizing the text style

The text style of the axis labels can be customized using the following options available in the [textStyle](#) property.

- [color](#) - It is used to change the text color of the axis labels.
- [fontFamily](#) - It is used to change the font family of the axis labels.
- [fontStyle](#) - It is used to change the font style of the axis labels.
- [fontWeight](#) - It is used to change the font weight of the axis labels.
- [size](#) - It is used to change the font size of the axis labels.
- [textAlignment](#) - It is used to position and align the axis labels. This property allows you to specify values such as **Near**, **Center**, and **Far**.

- [textOverflow](#) - When the axis label exceeds the intended space, this property is used to trim or wrap it. This property takes values such as **None**, **Trim**, and **Wrap**.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        'Month of August 2023',
        'Month of September 2023',
        'Month of October 2023',
        'Month of November 2023',
        'Month of December 2023'
    ],
    textStyle: {
        color: 'red',
        size: '15px',
        fontWeight: '650',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI',
        textAlignment: 'Center',
        textOverflow: 'Wrap'
    },
    opposedPosition: true
};
public yAxis: Object = {
    labels: [
        'Ace Apparels',
        'Alpha Apparels',
        'RL Garments',
        'RRD Garments',
        'RRD Apparels',
        'RR Garments',
        'SR Garments'
    ],
    textStyle: {
        color: 'red',
        size: '15px',
        fontWeight: '650',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI',
        textAlignment: 'Center',
        textOverflow: 'Wrap',
    },
    maxLabelLength: 70
};
public legendSettings: Object = {
    visible: false
};
public paletteSettings: Object = {
    palette: [
        { color: '#F0C27B' },
        { color: '#4B1248' }
    ],
};
}

```

**MAIN.TS**

```

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

```

### Providing line breaks

Axis labels with line breaks improve the readability of the HeatMap by splitting the text on an axis into multiple lines. The “`<br>`” character is used to add line breaks to the axis labels.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [1, 76],
    [19, 3]
  ];
  xAxis: Object = {
    labels: ['Actual<br>Accept', 'Actual<br>Reject'],
    opposedPosition: true
  };
  yAxis: Object = {
    labels: ['Actual<br>Accept', 'Actual<br>Reject'],
    maxLabelLength: 50
  };
}
```

### MAIN.TS

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

### Customizing labels when intersecting with other labels

When the axis labels intersect, [labelIntersectAction](#) property is used to handle the intersection. The `labelIntersectAction` property can take the following values.

- **None** - It specifies that no action is taken when the axis labels intersect.
- **Trim** - It specifies to trim the axis labels when they intersect.
- **Rotate45** - When the axis labels intersect, they are rotated to 45 degrees.

- **MultipleRows** - It specifies to show all the axis labels as multiple rows when they intersect.

The below example demonstrates to trim the axis labels by using the `labelIntersectAction` property.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [legendSettings]='legendSettings' [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public dataSource: Object[] = [
    [52, 65, 67, 45, 37, 52, 32],
    [68, 52, 63, 51, 30, 51, 51],
    [60, 50, 42, 53, 66, 70, 41],
    [66, 64, 46, 40, 47, 41, 45],
    [65, 42, 58, 32, 36, 44, 49],
    [54, 46, 61, 46, 40, 39, 41],
    [48, 46, 61, 47, 49, 41, 41],
    [69, 52, 41, 44, 41, 52, 46],
    [50, 59, 44, 43, 27, 42, 26],
    [47, 49, 66, 53, 50, 34, 31],
    [61, 40, 62, 26, 34, 54, 56]
  ];
  public titleSettings: Object = {
    text: 'Product wise Monthly sales revenue for a e-commerce website',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    enableTrim: true,
    labelIntersectAction: 'Trim',
    opposedPosition: true,
    labels: [
      'Month of Feburary 2023',
      'Month of March 2023',
      'Month of April 2023',
    ]
  };
}
```



```

        'Month of May 2023',
        'Month of June 2023',
        'Month of July 2023',
        'Month of August 2023',
        'Month of September 2023',
        'Month of October 2023',
        'Month of November 2023',
        'Month of December 2023'
    ]
};
public yAxis: Object = {
    enableTrim: true,
    labelIntersectAction: 'Trim',
    labels: [
        'Ace Apparels',
        'Alpha Apparels',
        'RL Garments',
        'RRD Garments',
        'RRD Apparels',
        'RR Garments',
        'SR Garments'
    ],
};
public legendSettings: Object = {
    visible: false
};
public paletteSettings: Object = {
    palette: [
        { color: '#F0C27B' },
        { color: '#4B1248' }
    ],
};
}

```

### MAIN.TS

```

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

```

### Rotating labels

The axis labels can be rotated to the desired angles by using the [labelRotation](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService],

```

```

standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[legendSettings]='legendSettings' [dataSource]='dataSource' [xAxis]='xAxis'
[yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public dataSource: Object[] = [
    [52, 65, 67, 45, 37, 52, 32],
    [68, 52, 63, 51, 30, 51, 51],
    [60, 50, 42, 53, 66, 70, 41],
    [66, 64, 46, 40, 47, 41, 45],
    [65, 42, 58, 32, 36, 44, 49],
    [54, 46, 61, 46, 40, 39, 41],
    [48, 46, 61, 47, 49, 41, 41],
    [69, 52, 41, 44, 41, 52, 46],
    [50, 59, 44, 43, 27, 42, 26],
    [47, 49, 66, 53, 50, 34, 31],
    [61, 40, 62, 26, 34, 54, 56]
  ];
  public titleSettings: Object = {
    text: 'Product wise Monthly sales revenue for a e-commerce website',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    labelRotation: 45,
    opposedPosition: true,
    labels: [
      'Month of Feburary 2023',
      'Month of March 2023',
      'Month of April 2023',
      'Month of May 2023',
      'Month of June 2023',
      'Month of July 2023',
      'Month of August 2023',
      'Month of September 2023',
      'Month of October 2023',
      'Month of November 2023',
      'Month of December 2023'
    ]
  };
  public yAxis: Object = {
    labelRotation: 45,
    labels: [
      'Ace Apparels',
      'Alpha Apparels',
      'RL Garments',
      'RRD Garments',
    ]
  }
}

```

```

        'RRD Apparels',
        'RR Garments',
        'SR Garments'
    ],
};
public legendSettings: Object = {
    visible: false
};
public paletteSettings: Object = {
    palette: [
        { color: '#F0C27B' },
        { color: '#4B1248' }
    ],
};
}

```

### MAIN.TS

```

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

```

### Label formatting

HeatMap supports formatting the axis labels by using the [labelFormat](#) property. Using this property, you can customize the axis label by global string format ('P', 'C', etc) or customized format like '{value}°C'.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [ LegendService, TooltipService, AdaptorService],
    standalone: true,
    selector: 'my-app',
    template:
        `

```

```

        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]
    ];
    xAxis: Object = {
        valueType: "DateTime",
        minimum: new Date(2007, 0, 1),
        intervalType: "Years",
        labelFormat: "yyyy",
    };
    yAxis: Object = {
        valueType: "Numeric",
        labelFormat: "${value}"
    };
    public legendSettings: Object = {
        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 intervals**

In HeatMap, you can define an interval between the axis labels using the [interval](#) property. In date-time axis, you can change the interval mode by using the [intervalType](#) property. The date-time axis supports the following interval types:

- Years
- Months
- Days
- Hours
- Minutes

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService, AdaptorService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ]
})

```

```

    ],
    providers: [ LegendService, TooltipService, AdaptorService],
    standalone: true,
    selector: 'my-app',
    template:
      `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[legendSettings]='legendSettings' [cellSettings]='cellSettings'
[titleSettings]='titleSettings'>
      </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object [] = [
      [36371, 25675, 28292, 33399, 35980, 38585, 39351, 39964, 36543,
30529, 33298, 36985],
      [34702, 27618, 31063, 34525, 36772, 35410, 38750, 39467, 35390,
34196, 35302, 35703],
      [34522, 31324, 32128, 34231, 36817, 34381, 37180, 38255, 32776,
32645, 31539, 32981],
      [32213, 28755, 29517, 31214, 33747, 33507, 35763, 36837, 32910,
33437, 30659, 31965],
      [31282, 28663, 32952, 33941, 34506, 36875, 38836, 35497, 34285,
34094, 32256, 33699],
      [31714, 29405, 33745, 32838, 33461, 35034, 36122, 37943, 34128,
30624, 32398, 33522],
      [32064, 28387, 33751, 32537, 34034, 35977, 37196, 38301, 33627,
34115, 31072, 33939],
      [32417, 27868, 30807, 33386, 35284, 36126, 39753, 40978, 35777,
35277, 31281, 35411],
      [32494, 29848, 34385, 35804, 37943, 38722, 41315, 41335, 37177,
37443, 32457, 37304],
      [34378, 29576, 30547, 35664, 36622, 38145, 40347, 41868, 38252,
36505, 29576, 36450],
      [35219, 31670, 32589, 34927, 36998, 39825, 41126, 42002, 37021,
36583, 32408, 37108]
    ];
    titleSettings: Object = {
      text: 'Monthly Flight Traffic at JFK Airport',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      valueType:"DateTime",
      minimum: new Date(2007,0,1),
      maximum: new Date(2017,0,1),
      intervalType:"Years",
      interval:2,
      labelFormat:"yyyy"
    };
    yAxis: Object = {
      labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May',
'Jun', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec'],

```

```

    };
    public legendSettings: Object = {
        visible: false,
    };
    public cellSettings: Object = {
        showLabel: false,
        border: {
            width: 0,
        },
        format: '{value} flights'
    };
}

```

### MAIN.TS

```

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

```

### Axis label increment

Axis label increment in the HeatMap is used to display the axis labels with regular interval values in numeric and date-time axes. The labels will be displayed with tick gaps when you set the label interval. But, to achieve the same behavior without tick gaps, use the label increment. You can set the axis label increment using the [increment](#) property and the default value of this property is 1.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object [] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],

```

```

[45, 7, 53, 81, 95, 79],
[60, 77, 74, 68, 88, 51],
[25, 25, 10, 12, 78, 14],
[25, 56, 55, 58, 12, 82],
[74, 33, 88, 23, 86, 59]];
xAxis: Object = {
  valueType: "Numeric",
  minimum: 0,
  increment: 2
};
yAxis: Object = {
  valueType: "Numeric",
  minimum: 0,
  increment: 3
};
}

```

### MAIN.TS

```

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

```

### Limiting labels in date-time axis

You can display the axis labels at specific time intervals along with the date-time axis using the [showLabelOn](#) property. This property supports the following types:

- None: Displays the axis labels based on the `intervalType` and `interval` property of the axis. This type is default value of the `showLabelOn` property.
- Years: Displays the axis labels on every year between given date-time range.
- Months: Displays the axis labels on every month between given date-time range.
- Days: Displays the axis labels on every day between given date-time range.
- Minutes: Displays the axis labels on every minute between given date-time range.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
import { Internationalization } from '@syncfusion/ej2-base';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService ],
  standalone: true,
  selector: 'my-app',
  template:

```

```

    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [legendSettings]='legendSettings' [cellSettings]='cellSettings'
[titleSettings]='titleSettings' [height]='height'
[paletteSettings]='paletteSettings' (tooltipRender)='tooltipRender($event)'>
    </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object [] = [
      [null, null, null, null, 16, 48, 0],
      [0, 15, 0, 24, 0, 39, 0],
      [0, 18, 37, 0, 0, 50, 0],
      [0, 10, 0, 0, 44, 5, 0],
      [0, 36, 0, 45, 20, 18, 0],
      [0, 28, 1, 42, 0, 10, 0],
      [0, 16, 32, 0, 1, 25, 0],
      [0, 31, 2, 9, 24, 0, 0],
      [0, 8, 47, 0, 0, 35, 0],
      [0, 31, 0, 0, 0, 40, 0],
      [0, 8, 0, 27, 0, 35, 0],
      [0, 12, 9, 45, 0, 8, 0],
      [0, 0, 13, 0, 22, 10, 0],
      [0, 16, 32, 0, 1, 25, 0],
      [0, 31, 2, 9, 24, 0, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 28, 14, 10, 0, 0, 0],
      [0, 36, 0, 45, 20, 18, 0],
      [0, 28, 1, 42, 0, 10, 0],
      [0, 31, 0, 24, 0, 40, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 36, 0, 45, 20, 18, 0],
      [0, 28, 1, 42, 0, 10, 0],
      [0, 31, 0, 24, 0, 40, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 36, 0, 45, 20, 18, 0],
      [0, 28, 1, 42, 0, 10, 0],
      [0, 31, 0, 24, 0, 40, 0],
      [0, 16, 32, 0, 1, 25, 0],
      [0, 31, 2, 9, 24, 0, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 10, 0, 36, 23, 19, 0],
      [0, 18, 37, 23, 0, 50, 0],
      [0, 28, 14, 10, 0, 0, 0],
      [0, 18, 37, 23, 0, 50, 0],
      [0, 18, 37, 23, 0, 50, 0],
      [0, 28, 14, 10, 0, 0, 0],
      [0, 31, 2, 9, 24, 0, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 10, 2, 0, 44, 5, 0],
      [0, 36, 0, 45, 20, 18, 0],
      [0, 28, 1, 42, 0, 10, 0],
      [0, 31, 0, 24, 0, 40, 1],
      [0, 16, 32, 0, 1, 25, 0],
      [0, 31, 2, 9, 24, 0, 0],
      [0, 8, 47, 27, 0, 35, 0],
      [0, 10, 2, 0, 44, 5, 0],
      [0, 12, 9, 45, 0, 8, 0],
      [0, 0, 13, 35, 22, 10, 0],
      [0, 28, 14, 10, 0, 0, 0],
      [0, 36, 0, 45, 20, 18, 2],
    ]
  }

```



```

[0, 28, 1, 42, 0, 10, 0],
[0, 31, 0, 24, 0, 40, 1],
[0, 8, 47, 27, 0, 35, 0],
[0, 10, 2, 0, 44, 5, 0],
[0, 31, 2, 9, 24, 0, 1],
[0, 8, 47, 27, 0, 35, 40],
[0, 10, 2, 0, 44, 5, null]
];
public height: String = '280px';
titleSettings: Object = {
    text: 'Annual Summary of User Activities in GitLab',
    textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    }
};
xAxis: Object = {
    opposedPosition: true,
    valueType: 'DateTime',
    minimum: new Date(2017, 6, 23),
    maximum: new Date(2018, 6, 30),
    intervalType: 'Days',
    showLabelOn: 'Months',
    labelFormat: 'MMM',
    increment: 7,
};
yAxis: Object = {
    labels: ['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'],
    isInversed: true,
};
public paletteSettings: Object = {
    palette: [
        { value: 0, color: 'rgb(238,238,238)', label: 'no
contributions' },
        { value: 1, color: 'rgb(172, 213, 242)', label: '1-15
contributions' },
        { value: 16, color: 'rgb(127, 168, 201)', label: '16-31
contributions' },
        { value: 32, color: 'rgb(82, 123, 160)', label: '31-49
contributions' },
        { value: 50, color: 'rgb(37, 78, 119)', label: '50+
contributions' },
    ],
    type: 'Fixed',
    emptyPointColor: 'white'
};
public legendSettings: Object = {
    position: 'Bottom',
    width: '20%',
    alignment: 'Near',
    showLabel: true,
    labelDisplayType: 'None',
    enableSmartLegend: true
};
public tooltipRender(args: ITooltipEventArgs): void {

```

```

        let intl: Internationalization = new Internationalization();
        let format: Function = intl.getDateFormat({ format: 'EEE MMM dd,
YYYY' });
        let newDate: Date = <Date>args.xValue;
        let date: Date = new Date(newDate.getTime());
        let axisLabel: string[] =
args.heatmap.axisCollections[1].axisLabels;
        let index: number = axisLabel.indexOf(args.yLabel);
        (date).setDate((date).getDate() + index);
        let value: string = format(date);
        args.content = [(args.value === 0 ? 'No' : args.value) + ' ' +
'contributions' + '<br>' + value];
    };
    public cellSettings: Object = {
        showLabel: false,
        border: {
            color: 'white'
        }
    };
}

```

### MAIN.TS

```

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

```

### Multilevel Labels

Multilevel labels are used to classify a group of axis labels as a single category, which is then displayed with a label. By using [multiLevelLabels](#), you can add multiple levels on top of the axis labels.

To divide and group the axis labels, you can use [multiLevelLabels](#) property. The starting and ending indexes of the axis labels can be set using the [start](#) and [end](#) properties in the [categories](#). The [text](#) property can be used to specify a name for the grouped axis labels.

The multilevel labels can be customized by using the following properties.

- [overflow](#) - It is used to trim or wrap the multilevel labels when the label overflows the intended space. NOTE: This property is only for x-axis.
- [alignment](#) - It is used to place and align the multilevel labels.
- [maximumTextWidth](#) - It is used to set the maximum width of the text. When the text length exceeds the maximum text width, the overflow action will be performed.
- [textStyle](#) - It is used to customize the font style of the multilevel labels.
- [border](#) - It is used to customize the border of the multilevel labels displayed in the x-axis and y-axis.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule, TooltipService } from '@syncfusion/ej2-angular-heatmap'

```

```

import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [52, 65, 67, 45, 37, 52, 32, 76, 60, 64, 82, 91],
    [68, 52, 63, 51, 30, 51, 51, 81, 70, 60, 88, 80],
    [60, 50, 42, 53, 66, 70, 41, 69, 76, 74, 86, 97],
    [66, 64, 46, 40, 47, 41, 45, 76, 83, 69, 92, 84],
    [65, 42, 58, 32, 36, 44, 49, 79, 83, 69, 83, 93],
    [54, 46, 61, 46, 40, 39, 41, 69, 61, 84, 84, 87],
    [48, 46, 61, 47, 49, 41, 41, 67, 78, 83, 98, 87],
    [69, 52, 41, 44, 41, 52, 46, 71, 63, 84, 83, 91],
    [50, 59, 44, 43, 27, 42, 26, 64, 76, 65, 81, 86],
    [47, 49, 66, 53, 50, 34, 31, 79, 78, 79, 89, 95],
    [61, 40, 62, 26, 34, 54, 56, 74, 83, 78, 95, 98]
  ];
  titleSettings: Object = {
    text: 'Product wise Monthly sales revenue for a e-commerce website',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Laptop', 'Mobile', 'Gaming', 'Cosmetics', 'Fragrance',
    'Watches', 'Handbags', 'Apparels', 'Kitchenware', 'Furniture', 'Home Decor'],
    border: { type: 'Rectangle', width:1, color: '#a19d9d' },
    multiLevelLabels: [
      {
        overflow: 'Trim',
        alignment: 'Near',
        textStyle: {
          color: 'black',
          fontWeight: 'Bold'
        },
      },
      border: { type: 'Rectangle', color: '#a19d9d' },
      categories: [
        { start: 0, end: 2, text: 'Electronics', },
        { start: 3, end: 4, text: 'Beauty and personal care',
maximumTextWidth: 50 },
        { start: 5, end: 7, text: 'Fashion', },
        { start: 8, end: 10, text: 'Household', }

```

```

    ]
  }
]
};
yAxis: Object = {
  labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'],
  multiLevelLabels: [
    {
      border: { type: 'Brace', color: '#a19d9d' },
      categories: [
        { start: 0, end: 2, text: 'Q1' },
        { start: 3, end: 5, text: 'Q2' },
        { start: 6, end: 8, text: 'Q3' },
        { start: 9, end: 11, text: 'Q4' }
      ]
    },
    {
      border: { type: 'Brace', color: '#a19d9d' },
      categories: [
        { start: 0, end: 5, text: 'First Half Yearly' },
        {
          start: 6,
          end: 11,
          text: 'Second Half Yearly'
        }
      ]
    },
    {
      border: { type: 'Brace', color: '#a19d9d' },
      categories: [
        { start: 0, end: 11, text: 'Yearly' }
      ]
    }
  ]
};
public paletteSettings: Object = {
  palette: [
    { color: '#F0C27B' },
    { color: '#4B1248' }
  ],
};
}

```

**MAIN.TS**

```

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

```

**Palette in Angular Heatmap chart component**

In heat map, each data point is displayed as a cell with applied color based on the data value. The palette in the heat map is used to define the color range for cells and gradient type for colors. You can

define the colors either in RGB or hex codes using the [color](#) property in `palette`. The defined colors are applied to the cell background based on the palette type and cell value.

### Palette types

You can display the heat map cells either in gradient colors or fixed colors.

#### Gradient

The smooth transition between the given palette colors can be applied for the heat map cells based on value. The heat map calculates all the gradient colors between the start and end colors for all distinct data values. Default start color and end color will be considered for gradient calculation, if the colors are not defined. The palette type must be defined as **Gradient** for the [type](#) property in `paletteSettings` property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  }
}
```

```

    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  public paletteSettings: Object = {
    palette: [
      { color: '#C06C84'},
      { color: '#6C5B7B'},
      { color: '#355C7D'}
    ],
    type: "Gradient"
  };
  public legendSettings: Object = {
    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));

```

### Fixed

In fixed palette type, solid colors are applied to the heat map cells. The data values can be grouped based on the number of colors defined for the heat map. The palette type should be defined as **Fixed** for the [type](#) property in the `paletteSettings` property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings'>
      </ejs-heatmap>`,

```

```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent{
    dataSource: Object[] = [
        [73, 39, 26, 39, 94, 0],
        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]];
    titleSettings: Object = {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    paletteSettings: Object = {
        palette: [
            { color: '#C06C84'},
            { color: '#6C5B7B'},
            { color: '#355C7D'}
        ],
        type: "Fixed"
    };
    legendSettings: Object = {
        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));

```

### Defining color stops

You can define the colors ranges or color stops for data values in both gradient and fixed palette types. You need to define the data value in the `value` property for `palette` property to calculate the color stops. The heat map automatically calculates the color stops if the `value` property is not defined. The `label` property is used to provide the additional information about the color that is to be displayed in the legend. If the label is not provided, the `value` is displayed in the legend. The labels can be automatically calculated based on data values, if both the values and labels are not defined.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings'>
      </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
```



```

        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    public paletteSettings: Object = {
        palette: [
            { color: '#C06C84', label: 'Low', value: 50 },
            { color: '#6C5B7B', label: 'Moderate', value: 80 },
            { color: '#355C7D', label: 'High', value: 100 }
        ],
        type: "Gradient"
    };
    public legendSettings: Object = {
        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));

```

See Also

- [How to enable smart legend](#)

### Legend in Angular Heatmap chart component

The legend is used to provide the information about the heat map cell. You can enable the legend by setting the [visible](#) property to true and injecting the `Legend` module using the `HeatMap.Inject(Legend)`.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings' [cellSettings]='cellSettings'>

```

```

</ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object[] = [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]
    ];
    titleSettings: Object = {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    public cellSettings: Object = {
      showLabel: false,
    };
    public paletteSettings: Object = {
      palette: [
        { value: 0, color: '#C2E7EC' },
        { value: 10, color: '#AEDFE6' },
        { value: 20, color: '#9AD7E0' },
        { value: 30, color: '#72C7D4' },
        { value: 40, color: '#5EBFCE' },
        { value: 50, color: '#4AB7C8' },
        { value: 60, color: '#309DAE' },
        { value: 70, color: '#2B8C9B' },
        { value: 80, color: '#206974' },
        { value: 90, color: '#15464D' },
        { value: 100, color: '#000000' },
      ],
    };
    public legendSettings: Object = {
      position: 'Right',
    };
  }

```

**MAIN.TS**

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

**Legend types**

Heat map supports two legend types: Gradient and list type.

- Gradient - This is a continuous color legend with smooth color transition between palette color values.
- List - List is a fixed color legend. Each palette color information is shown separately in the list item.

You can change the legend type by using the [type](#) property in the `paletteSettings` property.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
}
```

```

    titleSettings: Object = {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    cellSettings: Object = {
      showLabel: false,
    };
    paletteSettings: Object = {
      type: 'Fixed'
    };
    legendSettings: Object = {
      position: 'Right',
    };
  }

```

**MAIN.TS**

```

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

```

**Placement**

You can place the legend at left, right, top, or bottom to the heat map layout by using the [position](#) property. The legend is positioned at the right to the heat map by default.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:

```

```

    <ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [legendSettings]='legendSettings'>
    </ejs-heatmap>,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent{
    dataSource: Object[] = [
      [73, 39, 26, 39, 94, 0],
      [93, 58, 53, 38, 26, 68],
      [99, 28, 22, 4, 66, 90],
      [14, 26, 97, 69, 69, 3],
      [7, 46, 47, 47, 88, 6],
      [41, 55, 73, 23, 3, 79],
      [56, 69, 21, 86, 3, 33],
      [45, 7, 53, 81, 95, 79],
      [60, 77, 74, 68, 88, 51],
      [25, 25, 10, 12, 78, 14],
      [25, 56, 55, 58, 12, 82],
      [74, 33, 88, 23, 86, 59]];
    titleSettings: Object = {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    legendSettings: Object = {
      position: 'Top',
    };
  }
}

```

**MAIN.TS**

```

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

```

**Alignment**

You can align the legend as center, far, or near to the heat map using the [alignment](#) property.

**APP.COMPONENT.TS**

```

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

```

```

import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  legendSettings: Object = {
    position: 'Right',
    alignment: 'Near',
    height: '150px'
  };
}

```

**MAIN.TS**

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

### Legend dimensions

You can change the legend dimensions with values in pixels or percentage by using the [width](#) and [height](#) properties.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    legendSettings: Object = {
        position: 'Right',
        height: '150px',
    };
}

```

**MAIN.TS**

```

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

```

**Paging for legend**

Paging is available only for the list type legend in the heat map, and it can be enabled by default, when the legend items exceed the legend bounds. You can view each legend items by navigating between the pages using navigation buttons.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
[legendSettings]='legendSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],

```



```

[45, 7, 53, 81, 95, 79],
[60, 77, 74, 68, 88, 51],
[25, 25, 10, 12, 78, 14],
[25, 56, 55, 58, 12, 82],
[74, 33, 88, 23, 86, 59]
];

titleSettings: Object = {
  text: 'Sales Revenue per Employee (in 1000 US$)',
  textStyle: {
    size: '15px',
    fontWeight: '500',
    fontStyle: 'Normal',
    fontFamily: 'Segoe UI'
  }
};

xAxis: Object = {
  labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
};

yAxis: Object = {
  labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};

public paletteSettings: Object = {
  palette: [
    { value: 0, color: '#C2E7EC' },
    { value: 10, color: '#AEDFE6' },
    { value: 20, color: '#9AD7E0' },
    { value: 25, color: '#86CFDA' },
    { value: 30, color: '#72C7D4' },
    { value: 40, color: '#5EBFCE' },
    { value: 50, color: '#4AB7C8' },
    { value: 55, color: '#36AFC2' },
    { value: 60, color: '#309DAE' },
    { value: 70, color: '#2B8C9B' },
    { value: 75, color: '#257A87' },
    { value: 80, color: '#206974' },
    { value: 85, color: '#1B5761' },
    { value: 90, color: '#15464D' },
    { value: 100, color: '#000000' },
  ],
  type: 'Fixed'
};

public legendSettings: Object = {
  position: 'Right',
  height: "150px"
};
}

```

**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 legend feature, we need to inject `Legend` using `HeatMap.Inject(Legend)`.

### Smart Legend

Smart legend is another way of showing list type legend with responsiveness and readability, when the palette has more number of items. You can enable this smart legend by using the [enableSmartLegend](#) property when the palette type is set to **Fixed**.

In smart legend, you can change the display type of legend labels by using the [labelDisplayType](#) property.

The following are the legend label display types:

- All: Displays all labels in the legend.
- Edge: Displays the legend labels only at extreme ends.
- None: None of the labels are displayed. The tooltip will appear for this type of label display when hovering over the legend item.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
```

```

        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    cellSettings: Object = {
        showLabel: false,
    };
    paletteSettings: Object = {
        palette: [
            { value: 0, color: '#C2E7EC' },
            { value: 10, color: '#AEDFE6' },
            { value: 20, color: '#9AD7E0' },
            { value: 30, color: '#72C7D4' },
            { value: 40, color: '#5EBFCE' },
            { value: 50, color: '#4AB7C8' },
            { value: 60, color: '#309DAE' },
            { value: 70, color: '#2B8C9B' },
            { value: 80, color: '#206974' },
            { value: 90, color: '#15464D' },
            { value: 100, color: '#000000' },
        ],
        type: 'Fixed'
    };
    legendSettings: Object = {
        position: 'Bottom',
        width: '75%',
        enableSmartLegend: true
    };
}

```

**MAIN.TS**

```

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

```

**Legend Selection**

In the HeatMap, the legend selection is used to toggle the visibility of cell for viewing the specific range value. You can enable the legend selection using the [toggleVisibility](#) property.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        { value: 20, color: '#9AD7E0' },
        { value: 30, color: '#72C7D4' },
        { value: 40, color: '#5EBFCE' },
        { value: 50, color: '#4AB7C8' },
        { value: 60, color: '#309DAE' },
        { value: 70, color: '#2B8C9B' },
        { value: 80, color: '#206974' },
        { value: 90, color: '#15464D' },
        { value: 100, color: '#000000' },
      ],
      type: 'Fixed'
    };
    legendSettings: Object = {
      position: 'Bottom',
      width: '75%',
      enableSmartLegend: true,
      toggleVisibilty: 'true'
    };
  };
}

```

### MAIN.TS

```

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

```

### Legend Title

The legend title displays a specific information about the legend. You can enable the legend title by setting the [title](#) property by providing the text and customizing the legend title text style using the [textStyle](#) property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
    [legendSettings]='legendSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{

```

```

dataSource: Object[] = [
  [73, 39, 26, 39, 94, 0],
  [93, 58, 53, 38, 26, 68],
  [99, 28, 22, 4, 66, 90],
  [14, 26, 97, 69, 69, 3],
  [7, 46, 47, 47, 88, 6],
  [41, 55, 73, 23, 3, 79],
  [56, 69, 21, 86, 3, 33],
  [45, 7, 53, 81, 95, 79],
  [60, 77, 74, 68, 88, 51],
  [25, 25, 10, 12, 78, 14],
  [25, 56, 55, 58, 12, 82],
  [74, 33, 88, 23, 86, 59]
];

titleSettings: Object = {
  text: 'Sales Revenue per Employee'
};

xAxis: Object = {
  labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
};

yAxis: Object = {
  labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};

public cellSettings: Object = {
  showLabel: false,
};

public paletteSettings: Object = {
  palette: [
    { value: 0, color: '#6EB5D0' },
    { value: 50, color: '#7EDCA2' },
    { value: 100, color: '#DCD57E' },
  ],
};

public legendSettings: Object = {
  position: 'Right',
  title: {
    text: "1000 US$"
  }
};
}

```

**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 Heatmap chart component

## Cell customization

You can customize the cell by using the [cellSettings](#) property.

### Border

Change the width, color, and radius of the heat map cells by using the [border](#) property.

#### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  cellSettings: Object = {
    border: {

```

```

        width: 1,
        radius: 4,
        color: 'white'
    }
};
}

```

**MAIN.TS**

```

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

```

*Cell highlighting*

Enable or disable the cell highlighting while hovering over the heat map cells by using the [enableCellHighlighting](#) property.

Note: The cell highlighting only works in a SVG rendering mode.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {

```



```

        text: 'Sales Revenue per Employee (in US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    cellSettings: Object = {
        enableCellHighlighting: true
    };
}

```

**MAIN.TS**

```

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

```

*Color gradient mode*

The [colorGradientMode](#) property can be used to set the minimum and maximum values for colors based on row and column. Three types of color gradient modes are available.

- **Table:** The minimum and maximum value colors calculated for overall data.
- **Row:** The minimum and maximum value colors calculated for each row of data.
- **Column:** The minimum and maximum value colors calculated for each column of data.

Note: The default value of `colorGradientMode` is **Table**.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService],
    standalone: true,
    selector: 'my-app',
    template:

```

```

`<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [cellSettings]='cellSettings'
[paletteSettings]='paletteSettings'>
  </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  cellSettings: Object = {
    enableCellHighlighting: true
  };
  paletteSettings: Object = {
    colorGradientMode: 'Column'
  }
}

```

### MAIN.TS

```

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

```

### Background color

The background color of the heat map can be customized using the [backgroundColor](#) property.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    backgroundColor='#c7afcf' [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
}

```

**MAIN.TS**

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

### Margin

Set the margin for the heatmap from its container by using the [margin](#) property.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [margin]='margin'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  margin: Object = { left: 15, right: 15, top: 15, bottom: 15 };
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
```

```

        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

## MAIN.TS

```

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

```

## Title

The title is used to provide a quick information about the data plotted in heatmap. The [text](#) property is used to set the title for the heatmap. The text style of the title can be customized by using the [textStyle](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService, TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService, TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        [74, 33, 88, 23, 86, 59]];
    titleSettings: Object = {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Italic',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
  }
}

```

### MAIN.TS

```

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

```

### Data label

The visibility of data labels can be toggled using the [showLabel](#) property. By default, the data labels will be visible.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
`<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [cellSettings]='cellSettings'>
</ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],

```

```

        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]];
    titleSettings: Object = {
      text: 'Sales Revenue per Employee (in 1000 US$)',
      textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
      }
    };
    xAxis: Object = {
      labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
        'Mario'],
    };
    yAxis: Object = {
      labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    cellSettings: Object = {
      showLabel: false
    };
  }
}

```

**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 data label*

The label displayed in the heat map cell can be changed using the [cellRender](#) event.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ICellEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
})

```

```

standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' (cellRender)='cellRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  cellRender(args: ICellEventArgs): void {
    args.displayText = '$ ' + (args.value as number);
  };
}

```

### MAIN.TS

```

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

```

### Text style

The text attributes of the data label such as font-family, font-size, and color can be customized using the [textStyle](#) in the [cellSettings](#) property.



**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  cellSettings: Object = {
    textStyle: {
      fontStyle: 'Italic',
      fontFamily: 'Segoe UI'
    }
  }
}

```

```
};
}
```

## MAIN.TS

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

## Format

The format of the data label, such as currency, decimal, percent etc. can be changed using [format](#) property.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
```

```

        fontFamily: 'Segoe UI'
    }
};
xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
};
yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
cellSettings: Object = {
    format: '{value} K'
};
}

```

### MAIN.TS

```

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

```

### Template

Any HTML elements can be added as a template in the data labels by using the [labelTemplate](#) property of [cellSettings](#) in the HeatMap.

The following examples show various data binding methods in the HeatMap using the [labelTemplate](#) property.

### Array binding

By including `${value}` in the template content, the value from the data source for the corresponding cell can be displayed in the HeatMap cell as data label template content. Additionally, the x-axis and y-axis label values can be displayed by including `${xLabel}` and `${yLabel}` in the template content.

### Table

The following example demonstrates how to add a data label template for array table binding.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService, LegendService],
    standalone: true,
    selector: 'my-app',
    template:

```

```

`<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [cellSettings]='cellSettings'>
  </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [[4, 39], [3, 8], [1, 3], [1, 10], [4, 4], [2, 15]],
    [[4, 28], [5, 92], [5, 73], [3, 1], [3, 4], [4, 126]],
    [[4, 45], [5, 152], [0, 44], [4, 54], [5, 243], [2, 45]]];
  titleSettings: Object = {
    text: 'Commercial Aviation Accidents and Fatalities by year 2015 -
2017',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'inherit',
    },
  };
  xAxis: Object = {
    labels: ['2015', '2016', '2017']
  };
  yAxis: Object = {
    labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec']
  };
  cellSettings: Object = {
    labelTemplate:
      '<div style="width:25px;height:20px;text-align:center;padding-
top:2px;background-color:#5BBB9C; border: 1px solid #000000; border-
radius:50%;font-weight:bold;">${value}</div>'
  };
}

```

**MAIN.TS**

```

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

```

**Cell**

The following example demonstrates how to add a data label template for array cell binding.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, AdaptorService, LegendService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({

```

```

imports: [
    HeatMapModule
],
providers: [TooltipService, AdaptorService, LegendService],
standalone: true,
selector: 'my-app',
template:
`<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [cellSettings]='cellSettings'
[dataSourceSettings]='dataSourceSettings' >
    </ejs-heatmap>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    dataSource: Object[] = [
        [0, 0, [4, 39]], [0, 1, [3, 8]], [0, 2, [1, 3]], [0, 3, [1, 10]], [0,
4, [4, 4]], [0, 5, [2, 15]],
        [1, 0, [4, 28]], [1, 1, [5, 92]], [1, 2, [5, 73]], [1, 3, [3, 1]],
[1, 4, [3, 4]], [1, 5, [4, 126]],
        [2, 0, [4, 45]], [2, 1, [5, 152]], [2, 2, [0, 44]], [2, 3, [4, 54]],
[2, 4, [5, 243]], [2, 5, [2, 45]]
    ];
    titleSettings: Object = {
        text: 'Commercial Aviation Accidents and Fatalities by year 2015 -
2017',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'inherit',
        },
    };
    xAxis: Object = {
        labels: ['2015', '2016', '2017']
    };
    yAxis: Object = {
        labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
Dec']
    };
    cellSettings: Object = {
        labelTemplate:
`<div style="width:25px;height:20px;text-align:center;padding-
top:2px;background-color:#5BBB9C; border: 1px solid #000000; border-
radius:50%;font-weight:bold;">${value}</div>`
    };
    dataSourceSettings: Object = {
        isJsonData: false,
        adaptorType: 'Cell'
    };
}

```

**MAIN.TS**

```

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

```

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

### JSON binding

By including the desired field name in the template content, such as `${value}`, the value from the data source for the corresponding cell can be displayed in the HeatMap cell as data label template content.

### Table

The following example demonstrates how to add a data label template for JSON table binding.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, AdaptorService, LegendService } from
 '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, AdaptorService, LegendService],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
[cellSettings]='cellSettings' [dataSourceSettings]='dataSourceSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
  dataSource: Object[] = [
    {
      Year: '2017',
      image:
        'https://ej2.syncfusion.com/angular/demos/assets/circular-
        gauge/images/golf-ball.png',
      'Jan-Feb': [4, 39],
      'Mar-Apr': [3, 8],
      'May-Jun': [1, 3],
      'Jul-Aug': [1, 10],
      'Sep-Oct': [4, 4],
      'Nov-Dec': [2, 15],
    },
    {
      Year: '2016',
      image:
        'https://ej2.syncfusion.com/angular/demos/assets/circular-
        gauge/images/basket-ball.png',
      'Jan-Feb': [4, 28],
      'Mar-Apr': [5, 92],
      'May-Jun': [5, 73],
      'Jul-Aug': [3, 1],
      'Sep-Oct': [3, 4],
```

```

        'Nov-Dec': [4, 126],
    },
    {
        Year: '2015',
        image:
            'https://ej2.syncfusion.com/angular/demos/assets/circular-
            gauge/images/foot-ball.png',
        'Jan-Feb': [4, 45],
        'Mar-Apr': [5, 152],
        'May-Jun': [0, 44],
        'Jul-Aug': [4, 54],
        'Sep-Oct': [5, 243],
        'Nov-Dec': [2, 45],
    },
];
titleSettings: Object = {
    text: 'Commercial Aviation Accidents and Fatalities by year 2015 -
    2017',
    textStyle: {
        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'inherit',
    },
};
xAxis: Object = {
    labels: ['2015', '2016', '2017'],
};
yAxis: Object = {
    labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-
    Dec'],
};
cellSettings: Object = {
    labelTemplate:
        "<div><img style='width:20px;height:20px;' src='${image}'/>
</div>",
};
paletteSettings: Object = {
    palette: [{ color: '#C06C84' }, { color: '#6C5B7B' }, { color:
    '#355C7D' }],
    type: 'Gradient',
};
dataSourceSettings: Object = {
    isJsonData: true,
    adaptorType: 'Table',
    xDataMapping: 'Year',
};
}

```

## MAIN.TS

```

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

```

## Cell

The following example demonstrates how to add a data label template for JSON cell binding.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, AdaptorService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, AdaptorService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'
    [dataSourceSettings]='dataSourceSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    { Year: '2017', Months: 'Jan-Feb', Accidents: 4, Fatalities: 39 },
    { Year: '2017', Months: 'Mar-Apr', Accidents: 3, Fatalities: 8 },
    { Year: '2017', Months: 'May-Jun', Accidents: 1, Fatalities: 3 },
    { Year: '2017', Months: 'Jul-Aug', Accidents: 1, Fatalities: 10 },
    { Year: '2017', Months: 'Sep-Oct', Accidents: 4, Fatalities: 4 },
    { Year: '2017', Months: 'Nov-Dec', Accidents: 2, Fatalities: 15 },
    { Year: '2016', Months: 'Jan-Feb', Accidents: 4, Fatalities: 28 },
    { Year: '2016', Months: 'Mar-Apr', Accidents: 5, Fatalities: 92 },
    { Year: '2016', Months: 'May-Jun', Accidents: 5, Fatalities: 73 },
    { Year: '2016', Months: 'Jul-Aug', Accidents: 3, Fatalities: 1 },
    { Year: '2016', Months: 'Sep-Oct', Accidents: 3, Fatalities: 4 },
    { Year: '2016', Months: 'Nov-Dec', Accidents: 4, Fatalities: 126 },
    { Year: '2015', Months: 'Jan-Feb', Accidents: 4, Fatalities: 45 },
    { Year: '2015', Months: 'Mar-Apr', Accidents: 5, Fatalities: 152 },
    { Year: '2015', Months: 'May-Jun', Accidents: 0, Fatalities: 0 },
    { Year: '2015', Months: 'Jul-Aug', Accidents: 4, Fatalities: 54 },
    { Year: '2015', Months: 'Sep-Oct', Accidents: 5, Fatalities: 243 },
    { Year: '2015', Months: 'Nov-Dec', Accidents: 2, Fatalities: 45 }
  ];
  titleSettings: Object = {
    text: 'Commercial Aviation Accidents and Fatalities by year 2015 - 2017',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'inherit'
    }
  },
}
```



```

    };
    xAxis: Object = {
      labels: ['2015', '2016', '2017']
    };
    yAxis: Object = {
      labels: ['Jan-Feb', 'Mar-Apr', 'May-Jun', 'Jul-Aug', 'Sep-Oct', 'Nov-Dec']
    };
    cellSettings: Object = {
      labelTemplate: '<div> Accidents - ${Accidents}</div>'
    };
    dataSourceSettings: Object = {
      isJsonData: true,
      adaptorType: 'Cell',
      xDataMapping: 'Year',
      yDataMapping: 'Months',
      valueMapping: 'Fatalities'
    };
  };
}

```

**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

- [To customize the appearance of tool tip](#)

## Dimensions in Angular Heatmap chart component

### Size for container

Heat map can be rendered to its container size. You can set the size through inline or CSS.

`javascript

```
<div id='container'>
```

```
<div id='element' style="width:650px; height:350px;"></div>
```

```
</div>
```

,

### Size for heat map

You can set the size of heat map directly by using the [width](#) and [height](#) properties.

### In pixel

You can set the size for heat map in a pixel.

**APP.COMPONENT.TS**

```

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

```

```

import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [width]='width' [height]='height'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public width: String = '650px';
  public height: String = '350px';
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
}

```

**MAIN.TS**

```

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

```

### In percentage

By setting value in percentage, heat map gets its dimension with respect to its container. For example, when the height is '50%', heat map rendered to half of the container height.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [width]='width' [height]='height'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public width: String = '80%';
  public height: String = '90%';
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
    'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
```

```
};
}
```

## 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 Heatmap chart component

Tooltip is used to provide the details of the heat map cell, and this can be displayed, while hovering the cursor over the cell or performing tap action in touch devices.

### Default tooltip

You can enable the tooltip by setting the [showTooltip](#) property to true and injecting the `Tooltip` module using the `HeatMap.Inject(Tooltip)`.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'
    [showTooltip]='showTooltip'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
    [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
    [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
    [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
    [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
    [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
    [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
    [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]];
  titleSettings: Object = {
    text: 'Crude Oil Production of Non-OPEC Countries (in Million
    barrels per day)',
    textStyle: {
      size: '15px',
```

```

        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    }
};
xAxis: Object = {
    labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
'Russia', 'UK', 'USA']
};
yAxis: Object = {
    labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010'],
};
public cellSettings: Object = {
    showLabel: false,
};
public showTooltip: 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));

```

### Tooltip template

In heat map, you can customize the tooltip using the [tooltipRender](#) client-side event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
[titleSettings]='titleSettings' [cellSettings]='cellSettings'
(tooltipRender)= 'tooltipRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
    [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],

```

```

[2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
[3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
[3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
[3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
[5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
[6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]];
titleSettings: Object = {
  text: 'Crude Oil Production of Non-OPEC Countries (in Million
barrels per day)',
  textStyle: {
    size: '15px',
    fontWeight: '500',
    fontStyle: 'Normal',
    fontFamily: 'Segoe UI'
  }
};
xAxis: Object = {
  labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
'Russia', 'UK', 'USA']
};
yAxis: Object = {
  labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010'],
};
public cellSettings: Object = {
  showLabel: false,
};
public showTooltip: Boolean = true;
public tooltipRender(args: ITooltipEventArgs): void {
  args.content = ['In ' + args.yLabel + ', the ' + args.xLabel + '
produced ' + args.value + ' million barrels per day'];
};
}

```

### 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 appearance of Tooltip

The [fill](#) and [border](#) properties are used to customize the background color and border of the tooltip respectively. The [textStyle](#) property in the tooltip is used to customize the font of the tooltip text.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [

```

```

        HeatMapModule
    ],
    providers: [TooltipService],
    standalone: true,
    selector: 'my-app',
    template:
        `<ejs-heatmap id='container' style="display:block;"
        [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
        [titleSettings]='titleSettings' [cellSettings]='cellSettings'
        [paletteSettings]='paletteSettings' [tooltipSettings]='tooltipSettings'
        (tooltipRender)='tooltipRender($event)'>
        </ejs-heatmap>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent{
    dataSource: Object[] = [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]];
    titleSettings: Object = {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million
        barrels per day)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    xAxis: Object = {
        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
        'Russia', 'UK', 'USA']
    };
    yAxis: Object = {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
        '2007', '2008', '2009', '2010'],
    };
    public cellSettings: Object = {
        showLabel: false,
    };
    public paletteSettings: Object = {
        palette: [
            { color: '#F0ADCE'},
            { color: '#19307B'}],
    },
    };
    public tooltipSettings: Object = {
        fill: '#696295',
        textStyle: {
            color: 'FFFFFF',
            size: '12px'
        },
    },
}

```

```

        border: {
            width: 2,
            color: '#F0C27B'
        }
    };
    public showTooltip: Boolean = true;
    public tooltipRender(args: ITooltipEventArgs): void {
        args.content = ['In ' + args.yLabel + ', the ' + args.xLabel + '
produced ' + args.value + ' million barrels per day'];
    };
}

```

## 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 tooltip feature, we need to inject `Tooltip` using `HeatMap.Inject(Tip)`.

## Selection in Angular HeatMap chart component

In the HeatMap, the cell selection is used to select single or multiple HeatMap cells at runtime and get the selected cell details using the [cellSelected](#) event. You can enable the cell selection using the [allowSelection](#) property.

The HeatMap cells can be selected using the following interactions, as shown in the table below.

Modes of Interactions	Description
Mouse	HeatMap cells can be selected by clicking or dragging and dropping over them.
Touch	HeatMap cells can be selected by tapping or dragging and dropping over them.
Keyboard	The <b>Ctrl</b> key on the keyboard can be used to enable multiple cell selection with mouse and touch interaction. The <b>Ctrl</b> key can only be used if the <code>enableMultiSelect</code> property is set to <b>true</b> in order to enable multiple cell selection.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap';
import { TooltipService } from '@syncfusion/ej2-angular-heatmap';
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService],
})

```



```

standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' [allowSelection]='allowSelection'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  allowSelection: 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));

```

### Enable single cell selection

In the HeatMap, the [enableMultiSelect](#) property is used to allow single cell selection. When you set the [enableMultiSelect](#) property to **false**, only one cell is selected. By default, [enableMultiSelect](#) property is set to **true**.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [allowSelection]='allowSelection'
    [enableMultiSelect]='enableMultiSelect'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
    'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
  allowSelection: boolean = true;
  enableMultiSelect: 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));
```

*Clearing cell selection*

The [clearSelection](#) method can be used to clear all the selected cells. The below example illustrates the same.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { HeatMapComponent } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService],
  standalone: true,
  selector: 'my-app',
  template:
    `

```

```

        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    public xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    public yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

## 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 HeatMap chart component

The HeatMap component follows commonly used accessibility guidelines and standards, such as [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#).

The accessibility compliance for the HeatMap component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device 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

HeatMap component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the HeatMap component.

Attributes	Purpose
------------	---------

---	---
-----	-----

<code>role=img</code>	It is specified in the legend and border of the HeatMap. This role is provided to specify the information in a visual manner.
-----------------------	---

<code>role=region</code>	It specifies the HeatMap areas that do not support interactive functions like cell selection.
--------------------------	---

<code>aria-label</code>	Provides an accessible name for the title, legend title, legend item labels, axis labels, cell labels and multilevel labels.
-------------------------	--

### Screen reading in HeatMap

HeatMap has built-in accessibility features like screen reading. Screen reading in the HeatMap component allows all users, regardless of ability or disability, to use the component. The following HeatMap elements will be read aloud with screen reading software like Narrator for Windows.

Elements	Description
----------	-------------

---	---
-----	-----

Title	Reads the contents of the HeatMap chart's title.
-------	--

Axis labels	Reads the x and y axis labels of the HeatMap chart.
-------------	---

Multilevel labels	Reads the multilevel labels in the x and y axis of the HeatMap chart.
-------------------	---

Cell labels	Reads the labels from the cells in the Heatmap chart.
-------------	---

Legend title	Reads the contents of the legend's title as specified in HeatMap chart.
--------------	---

Legend item label	Reads the label of a legend item in HeatMap chart.
-------------------	--

### Ensuring accessibility

The HeatMap component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the HeatMap component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the HeatMap component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

## Events in Angular Heatmap chart component

This section describes the HeatMap chart event, which occurs when the required actions are performed.

### cellClick

When you click on a HeatMap cell, the [cellClick](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ICellClickEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (cellClick)=(cellClick($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public cellClick(args: ICellClickEventArgs): void {
    console.log("The cell click event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
```

```

        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]]];
    public titleSettings: Object = {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    public xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    public yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

### MAIN.TS

```

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

```

### cellDoubleClick

When you double click on a HeatMap cell, the [cellDoubleClick](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ICellClickEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService, LegendService],
    standalone: true,
    selector: 'my-app',
    template:
        `<ejs-heatmap id='container'
        (cellDoubleClick)=(cellDoubleClick($event)) style="display:block;"
        [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
        [titleSettings]='titleSettings'>
        </ejs-heatmap>`,

```

```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent {
        public cellDoubleClick(args: ICellClickEventArgs): void {
            console.log("The cell double click event has been triggered!!!");
        }
        public dataSource: Object[] = [
            [73, 39, 26, 39, 94, 0],
            [93, 58, 53, 38, 26, 68],
            [99, 28, 22, 4, 66, 90],
            [14, 26, 97, 69, 69, 3],
            [7, 46, 47, 47, 88, 6],
            [41, 55, 73, 23, 3, 79],
            [56, 69, 21, 86, 3, 33],
            [45, 7, 53, 81, 95, 79],
            [60, 77, 74, 68, 88, 51],
            [25, 25, 10, 12, 78, 14],
            [25, 56, 55, 58, 12, 82],
            [74, 33, 88, 23, 86, 59]];
        public titleSettings: Object = {
            text: 'Sales Revenue per Employee (in 1000 US$)',
            textStyle: {
                size: '15px',
                fontWeight: '500',
                fontStyle: 'Normal',
                fontFamily: 'Segoe UI'
            }
        };
        public xAxis: Object = {
            labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
                'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
        };
        public yAxis: Object = {
            labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
        };
    }
}

```

## MAIN.TS

```

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

```

## cellRender

The [cellRender](#) event will be triggered before each HeatMap cell is rendered. To know more about arguments of this event, refer [here](#).

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'

```



```

import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ICellEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (cellRender)=(cellRender($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public cellRender(args: ICellEventArgs): void {
    console.log("The cell render event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  public yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
}

```

**MAIN.TS**

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

### cellSelected

When single or multiple cells in the HeatMap are selected, the [cellSelected](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ISelectedEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (cellSelected)=(cellSelected($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings' [allowSelection]='allowSelection'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public cellSelected(args: ISelectedEventArgs): void {
    console.log("The cell selected event has been triggered!!!");
  }
  public allowSelection: boolean = true;
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
```

```

        size: '15px',
        fontWeight: '500',
        fontStyle: 'Normal',
        fontFamily: 'Segoe UI'
    }
};
public xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
};
public yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
};
}

```

### MAIN.TS

```

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

```

created

Once HeatMap has been completely rendered, the [created](#) event is triggered.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (created)=(created($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public created(args: Object): void {
    console.log("The created event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],

```

```

        [93, 58, 53, 38, 26, 68],
        [99, 28, 22, 4, 66, 90],
        [14, 26, 97, 69, 69, 3],
        [7, 46, 47, 47, 88, 6],
        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]];
    public titleSettings: Object = {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    public xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    public yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

**MAIN.TS**

```

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

```

**legendRender**

The [legendRender](#) event is triggered before the legend is rendered. To know more about arguments of this event, refer [here](#).

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ILegendRenderEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
    imports: [
        HeatMapModule
    ],
}

```

```

providers: [TooltipService, LegendService],
standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (legendRender)=(legendRender($event))
style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
[yAxis]='yAxis'
  [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public legendRender(args: ILegendRenderEventArgs): void {
    console.log("The legend render event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  public yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
}

```

**MAIN.TS**

```

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

```

load

The [load](#) event is triggered before the HeatMap is rendered. To know more about arguments of this event, refer [here](#).

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ILoadedEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (load)=(load($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public load(args: ILoadedEventArgs): void {
    console.log("The load event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
```

```

        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    public yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

### MAIN.TS

```

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

```

### loaded

Once HeatMap is loaded, the [loaded](#) event is triggered. To know more about arguments of this event, refer [here](#).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ILoadedEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (loaded)=(loaded($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public loaded(args: ILoadedEventArgs): void {
    console.log("The loaded event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
  ]
}

```

```

        [41, 55, 73, 23, 3, 79],
        [56, 69, 21, 86, 3, 33],
        [45, 7, 53, 81, 95, 79],
        [60, 77, 74, 68, 88, 51],
        [25, 25, 10, 12, 78, 14],
        [25, 56, 55, 58, 12, 82],
        [74, 33, 88, 23, 86, 59]];
    public titleSettings: Object = {
        text: 'Sales Revenue per Employee (in 1000 US$)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    public xAxis: Object = {
        labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
            'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
    };
    public yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
}

```

### MAIN.TS

```

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

```

### resized

When the window is resized, the [resized](#) event is triggered to notify the resize of the HeatMap. To know more about arguments of this event, refer [here](#).

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, IResizeEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
    imports: [
        HeatMapModule
    ],
    providers: [TooltipService, LegendService],
    standalone: true,
    selector: 'my-app',
    template:

```



```

`<ejs-heatmap id='container' (resized)=(resized($event))
style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
[yAxis]='yAxis'
  [titleSettings]='titleSettings'>
  </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public resized(args: IResizeEventArgs): void {
    console.log("The resized event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  public yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
}

```

**MAIN.TS**

```

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

```

**tooltipRender**

The [tooltipRender](#) event is triggered before the tooltip is rendered on the HeatMap cell. To know more about arguments of this event, refer [here](#).

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Tooltip, Legend, ITooltipEventArgs } from '@syncfusion/ej2-angular-heatmap';
HeatMap.Inject(Tooltip, Legend);
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'my-app',
  template:
    `<ejs-heatmap id='container' (tooltipRender)=(tooltipRender($event))
    style="display:block;" [dataSource]='dataSource' [xAxis]='xAxis'
    [yAxis]='yAxis'
    [titleSettings]='titleSettings'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public tooltipRender(args: ITooltipEventArgs): void {
    console.log("The tooltip render event has been triggered!!!");
  }
  public dataSource: Object[] = [
    [73, 39, 26, 39, 94, 0],
    [93, 58, 53, 38, 26, 68],
    [99, 28, 22, 4, 66, 90],
    [14, 26, 97, 69, 69, 3],
    [7, 46, 47, 47, 88, 6],
    [41, 55, 73, 23, 3, 79],
    [56, 69, 21, 86, 3, 33],
    [45, 7, 53, 81, 95, 79],
    [60, 77, 74, 68, 88, 51],
    [25, 25, 10, 12, 78, 14],
    [25, 56, 55, 58, 12, 82],
    [74, 33, 88, 23, 86, 59]];
  public titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  public xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',
      'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin', 'Mario'],
  };
  public yAxis: Object = {
    labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
  };
};

```

```
}

```

**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****Tooltip template in Angular Heatmap chart component**

You can show a tooltip as a table using the `template` property in `tooltipSettings`.

The following steps describe how to show the table tooltip.

**Step 1:** Initialize the tooltip template div as shown in the following html page.

```
`html
```

```
<script id="tooltipTemplate" type="text/x-template">
```

```
<div id='templateWrap'>
```

<code>\${xValue}:</code>	<code>\${yValue}</code>	<code>\${value}</code>
--------------------------	-------------------------	------------------------

```
</div>
```

```
</script>
```

```
`
```

**Step 2:** Set the element id to the `template` property in `tooltipSettings` to show the tooltip template.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { TooltipService, LegendService } from '@syncfusion/ej2-angular-heatmap'
import { Component, ViewEncapsulation } from '@angular/core';
import { HeatMap, Legend, Tooltip } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [TooltipService, LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-heatmap id='container' style="display:block;"
    [dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
    [titleSettings]='titleSettings' [cellSettings]='cellSettings'
    [showTooltip]='showTooltip' [tooltipSettings]='tooltipSettings'
    [legendSettings]='legendSettings' >
    </ejs-heatmap>`,
```

```

        encapsulation: ViewEncapsulation.None
    })
    export class AppComponent{
    dataSource: Object[] = [
        [0.72, 0.71, 0.71, 0.67, 0.72, 0.53, 0.53, 0.56, 0.58, 0.56],
        [2.28, 2.29, 2.09, 1.84, 1.64, 1.49, 1.49, 1.39, 1.32, 1.23],
        [2.02, 2.17, 2.30, 2.39, 2.36, 2.52, 2.62, 2.57, 2.57, 2.74],
        [3.21, 3.26, 3.45, 3.47, 3.42, 3.34, 3.14, 2.83, 2.64, 2.61],
        [3.22, 3.13, 3.04, 2.95, 2.69, 2.49, 2.27, 2.18, 2.06, 1.87],
        [3.30, 3.39, 3.40, 3.48, 3.60, 3.67, 3.73, 3.79, 3.79, 4.07],
        [5.80, 5.74, 5.64, 5.44, 5.18, 5.08, 5.07, 5.00, 5.35, 5.47],
        [6.91, 7.40, 8.13, 8.80, 9.04, 9.24, 9.43, 9.35, 9.49, 9.69]];
    titleSettings: Object = {
        text: 'Crude Oil Production of Non-OPEC Countries (in Million
barrels per day)',
        textStyle: {
            size: '15px',
            fontWeight: '500',
            fontStyle: 'Normal',
            fontFamily: 'Segoe UI'
        }
    };
    tooltipSettings: Object = {
        template: '#tooltipTemplate'
    };
    xAxis: Object = {
        labels: ['Canada', 'China', 'Egypt', 'Mexico', 'Norway',
'Russia', 'UK', 'USA']
    };
    yAxis: Object = {
        labels: ['2000', '2001', '2002', '2003', '2004', '2005', '2006',
'2007', '2008', '2009', '2010'],
    };
    public cellSettings: Object = {
        showLabel: false,
    };
    public legendSettings: Object = {
        position: 'Right',
    };
    public showTooltip: 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));

```

### Legend customization in Angular Heatmap chart component

You can change the legend label using the `legendRender` client-side event. You can also hide the legend label using this client-side event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { HeatMapModule } from '@syncfusion/ej2-angular-heatmap'
import { LegendService } from '@syncfusion/ej2-angular-heatmap'
import { EmitType } from '@syncfusion/ej2-base';
import { Component, ViewEncapsulation } from '@angular/core';
import { ILegendRenderEventArgs } from '@syncfusion/ej2-angular-heatmap';
@Component({
  imports: [
    HeatMapModule
  ],
  providers: [ LegendService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-heatmap id='container' style="display:block;"
[dataSource]='dataSource' [xAxis]='xAxis' [yAxis]='yAxis'
  [titleSettings]='titleSettings' [paletteSettings]='paletteSettings'
[legendSettings]='legendSettings' [cellSettings]='cellSettings'
(legendRender)='legendRender($event)'>
    </ejs-heatmap>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent{
  dataSource: Object[] = [
    [73000, 39000, 26000, 39000, 94000, 0],
    [93000, 58000, 53000, 38000, 26000, 68000],
    [99000, 28000, 22000, 4000, 66000, 9000],
    [14000, 26000, 97000, 69000, 69000, 3000],
    [7000, 46000, 47000, 47000, 88000, 6000],
    [41000, 55000, 73000, 23000, 30000, 79000],
    [56000, 69000, 21000, 86000, 3000, 33000],
    [45000, 7000, 53000, 81000, 95000, 79000],
    [60000, 77000, 74000, 68000, 88000, 51000],
    [25000, 25000, 10000, 12000, 78000, 14000],
    [25000, 56000, 55000, 58000, 12000, 82000],
    [74000, 33000, 88000, 23000, 86000, 59000]];
  legendSettings: any;
  public legendRender(args: ILegendRenderEventArgs | any): void {
    if(args.text=='25,000' || args.text=='50,000' || args.text=='99,000'){
      args.text = args.text.replace(/,/g, "");
      args.text = `${parseInt((args.text/1000 as any))}` + "k "+"$";
    } else {
      args.cancel=true;
    }
  };
  titleSettings: Object = {
    text: 'Sales Revenue per Employee (in 1000 US$)',
    textStyle: {
      size: '15px',
      fontWeight: '500',
      fontStyle: 'Normal',
      fontFamily: 'Segoe UI'
    }
  };
  xAxis: Object = {
    labels: ['Nancy', 'Andrew', 'Janet', 'Margaret', 'Steven',

```

```

        'Michael', 'Robert', 'Laura', 'Anne', 'Paul', 'Karin',
'Mario'],
    };
    yAxis: Object = {
        labels: ['Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat'],
    };
    cellSettings: Object = {
        showLabel: false,
    };
    paletteSettings: Object = {
        palette: [
            { value: 0, color: '#C2E7EC' },
            { value: 25000, color: '#AEDFE6' },
            { value: 50000, color: '#9AD7E0' },
            { value: 75000, color: '#72C7D4' },
            { value: 99000, color: '#5EBFCE' },
        ],
        type: 'Gradient'
    };
}

```

## 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 Heatmap chart component

This article describes the API migration process of heat map component from Essential JS 1 to Essential JS 2.

### Members

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Specifies the width of the heat map | **Property:** *width*  
`<ej-heatmap id="HeatMap" [width]="100%"></ej-heatmap>` | **Property:** *width*  
`<ejs-heatmap id='container' [width]='width'></ejs-heatmap>` `<br><br><code>this.width = '300'</code>` |

| Specifies the height of the heat map | **Property:** *height*  
`<ej-heatmap id="HeatMap" [height]="100%"></ej-heatmap>` | **Property:** *height*  
`<ejs-heatmap id='container' [height]='height'></ejs-heatmap>` `<br><br><code>this.height = '300'</code>` |

| Enables or disables tooltip of heat map | **Property:** *showtooltip*  
`<ej-heatmap id="HeatMap" [showTooltip]="true"></ej-heatmap>` | **Property:** *showTooltip*  
`<ejs-heatmap id='container' [showTooltip]='showTooltip'></ejs-heatmap>` `<br><br><code>this.showTooltip = true</code>` |

| Defines the tooltip that should be shown when the mouse hovers over cells. | **Property:**

*tooltipSettings.templateId*  
`<ej-heatmap id="HeatMap" [tooltipSettings]="tooltipSettings"></ej-heatmap>`  
`<ej-heatmap id='container' (tooltipRender)= 'tooltipRender($event)'></ej-heatmap>`  
`<ej-heatmap id='container' (tooltipRender)= 'tooltipRender($event)'></ej-heatmap>`  
`<ej-heatmap id='container' (tooltipRender)= 'tooltipRender($event)'></ej-heatmap>`

| Specifies the source data of the heat map. | **Property:** *itemsSource*  
`<ej-heatmap id="HeatMap" [itemsSource]="itemsSource"></ej-heatmap>`

`<ej-heatmap id='container' [dataSource]='dataSource'></ej-heatmap>`  
`<ej-heatmap id='container' [dataSource]='dataSource'></ej-heatmap>`  
`<ej-heatmap id='container' [dataSource]='dataSource'></ej-heatmap>`

| Specifies whether the cell content can be visible or not. | **Property:**

*heatmapCell.showContent*  
`<ej-heatmap id="HeatMap" [heatmapCell]="heatmapCell"></ej-heatmap>`  
`<ej-heatmap id='container' [cellSettings]='cellSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [cellSettings]='cellSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [cellSettings]='cellSettings'></ej-heatmap>`

| Specifies the color of the heat map column data. | **Property:**

*colorMappingCollection.color*  
`<ej-heatmap><e-colormappingcollection><e-colormapping color="#8ec8f8"></e-colormapping><e-colormapping color="#0d47a1"></e-colormapping></e-colormappingcollection></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`

| Specifies the color values of the heat map column data. | **Property:**

*colorMappingCollection.value*  
`<ej-heatmap><e-colormappingcollection><e-colormapping [value]="0"></e-colormapping><e-colormapping [value]="100"></e-colormapping></e-colormappingcollection></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`

| Specifies the label text of the heat map color. | **Property:**

*colorMappingCollection.label.text*  
`<ej-heatmap><e-colormappingcollection><e-colormapping [label]="label"></e-colormapping></e-colormappingcollection></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`  
`<ej-heatmap id='container' [paletteSettings]='paletteSettings'></ej-heatmap>`

| Specifies the style of the heat map color label. | **Property:** *colorMappingCollection.label.bold* **Property:**

*colorMappingCollection.label.italic*  
`<ej-heatmap><e-colormappingcollection><e-colormapping [label]="labelBold"></e-colormapping><e-colormapping [label]="labelItalic"></e-colormapping></e-colormappingcollection></ej-heatmap>`

`colormappingcollection><br></ej-heatmap><br><br><code>this.labelBold = { bold: true  
</code>this.labelItalic = { italic: true }</code>| Property:  
legendSettings.textStyle.fontSize<br><br><code><ej-heatmap id='container'  
[legendSettings]='legendSettings'><br></ej-heatmap><br><br><code>this.legendSettings =  
{textStyle: { fontStyle:'bold' }}</code>|`

| Specifies the font size of the heat map label. | **Property:**  
`colorMappingCollection.label.fontSize<br><br><code><ej-heatmap><br><e-  
colormappingcollection><br><e-colormapping [label]="label"><br></e-colormapping><br></e-  
colormappingcollection><br></ej-heatmap><br><br><code>this.label = { fontSize: 18 }</code>|  
Property: legendSettings.textStyle.size<br><br><code><ej-heatmap id='container'  
[legendSettings]='legendSettings'><br></ej-heatmap><br><br><code>this.legendSettings =  
{textStyle: { size: 18 }}</code>|`

| Specifies the font family of the heat map label. | **Property:**  
`colorMappingCollection.label.fontFamily<br><br><code><ej-heatmap><br><e-  
colormappingcollection><br><e-colormapping [label]="label"><br></e-colormapping><br></e-  
colormappingcollection><br></ej-heatmap><br><br><code>this.label = { fontFamily: "Arial"  
</code>| Property: legendSettings.textStyle.fontFamily<br><br><code><ej-heatmap  
id='container' [legendSettings]='legendSettings'><br></ej-heatmap><br>  
<br><code>this.legendSettings = {textStyle: { fontFamily: 'Arial' }}</code>|`

| Specifies the font color of the heat map label. | **Property:**  
`colorMappingCollection.label.fontColor<br><br><code><ej-heatmap><br><e-  
colormappingcollection><br><e-colormapping [label]="label"><br></e-colormapping><br></e-  
colormappingcollection><br></ej-heatmap><br><br><code>this.label = { fontColor: "red" }</code>|  
Property: legendSettings.textStyle.fontFamily<br><br><code><ej-heatmap id='container'  
[legendSettings]='legendSettings'><br></ej-heatmap><br><br><code>this.legendSettings =  
{textStyle: { color: 'red' }}</code>|`

| Specifies the mapping name of the column. | **Property:**  
`itemsMapping.column.propertyName<br><br><code><ej-heatmap  
[itemsMapping]="itemsMapping"><br></ej-heatmap><br><br><code>this.itemsMapping =  
{column: { "propertyName": "ProductName" }}</code>| Property:  
dataSource.yDataMapping<br><br><code><ej-heatmap id='container'  
[dataSource]='dataSource'><br></ej-heatmap><br><br><code>this.dataSource = {data:  
heatmapData,yDataMapping: 'columnid'}</code>|`

| Specifies the mapping name of the row. | **Property:**  
`itemsMapping.row.propertyName<br><br><code><ej-heatmap  
[itemsMapping]="itemsMapping"><br></ej-heatmap><br><br><code>this.itemsMapping = {row: {  
"displayName": "Product Name" }}</code>| Property:  
dataSource.xDataMapping<br><br><code><ej-heatmap id='container'  
[dataSource]='dataSource'><br></ej-heatmap><br><br><code>this.dataSource = {data:  
heatmapData,xDataMapping: 'rowid'}</code>|`

| Specifies the mapping name of the row. | **Property:**  
`itemsMapping.value.displayName<br><br><code><ej-heatmap  
[itemsMapping]="itemsMapping"><br></ej-heatmap><br><br><code>this.itemsMapping = {value:`



```
{ "displayName": "Product Name" }}</code>| Property:  
dataSource.valueMapping<br/><br/><code><ejs-heatmap id='container'  
[dataSource]='dataSource'><br/></ejs-heatmap><br> <br><code>this.dataSource = {data:  
heatmapData,valueMapping: 'value'}</code>|
```

### Events

```
<!-- markdownlint-disable MD033 -->
```

```
| Behavior | API in Essential JS 1 | API in Essential JS 2 |
```

```
| --- | --- | --- |
```

```
| Triggered when the cell get clicked. | Property: cellSelected<br/><br/><code><ej-heatmap  
(actionComplete)="actionComplete($event)"><br/></ej-heatmap><br>  
<br><code>this.actionComplete = function(args) {}</code>| Property: cellClick<br/><br/><code><ejs-  
heatmap id='container' (cellClick)="cellClick($event)"><br/></ejs-heatmap><br><br><code>  
this.cellClick = function(args) {}</code>|
```

## Image Editor

### Getting started with Angular Image editor component

This section explains how to create and demonstrate the basic usage of the [Angular Image EditorLink to the Video](#) module.

To get started quickly with angular Image Editor component using angular CLI, you can check the video below.

### Dependencies

The list of dependencies required to use the Image Editor module in your application is given below:

```
`javascript  
|-- @syncfusion/ej2-angular-image-editor  
|-- @syncfusion/ej2-angular-base  
|-- @syncfusion/ej2-base  
-- @syncfusion/ej2-data  
|-- @syncfusion/ej2-buttons  
-- @syncfusion/ej2-lists  
|-- @syncfusion/ej2-image-editor  
|-- @syncfusion/ej2-dropdowns  
|-- @syncfusion/ej2-inputs  
|-- @syncfusion/ej2-navigations  
|-- @syncfusion/ej2-popups  
|-- @syncfusion/ej2-splitbuttons  
,
```

### 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 Image Editor package

To install Image Editor package, use the following command.

`

```
npm install @syncfusion/ej2-angular-image-editor --save
```

`

The above package installs [Image Editor dependencies](#) which are required to render the component in the Angular environment.

### Adding Image Editor module

Import Image Editor module into Angular application(app.module.ts) from the package `@syncfusion/ej2-angular-image-editor`.

```
`typescript
```

```
import { NgModule } from '@angular/core';
```

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

```
// Importing ImageEditorModule from ej2-angular-image-editor package.
```

```
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor';
```

```
import { AppComponent } from './app.component';
```

```
@NgModule({
```

```
  imports: [ BrowserModule, ImageEditorModule ], // Declaration of ImageEditor module into NgModule.
```

```
  declarations: [ AppComponent ],
```

```
  bootstrap: [ AppComponent ]
```

```
})
```

```
export class AppModule { }
```

### Adding Syncfusion Image Editor component

Modify the template in `app.component.ts` file to render the Angular Image Editor component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: `<!-- To render Image Editor. -->
<div id="wrapperDiv" style="width:550px;height:350px;">
<ejs-imageeditor></ejs-imageeditor>
</div>`
})
export class AppComponent { }
```

### Adding CSS reference

Add Angular Image Editor 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-lists/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-navigations/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-image-editor/styles/material.css";
```

### Running the application

Run the application in the browser using the following command:

```
ng serve
```

The following example shows a basic Image Editor component.

### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component } from '@angular/core';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor></ejs-imageeditor>
    </div>
  </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));
```

You can also explore our [Angular Image Editor example](#) that shows how to render the Image Editor in Angular.

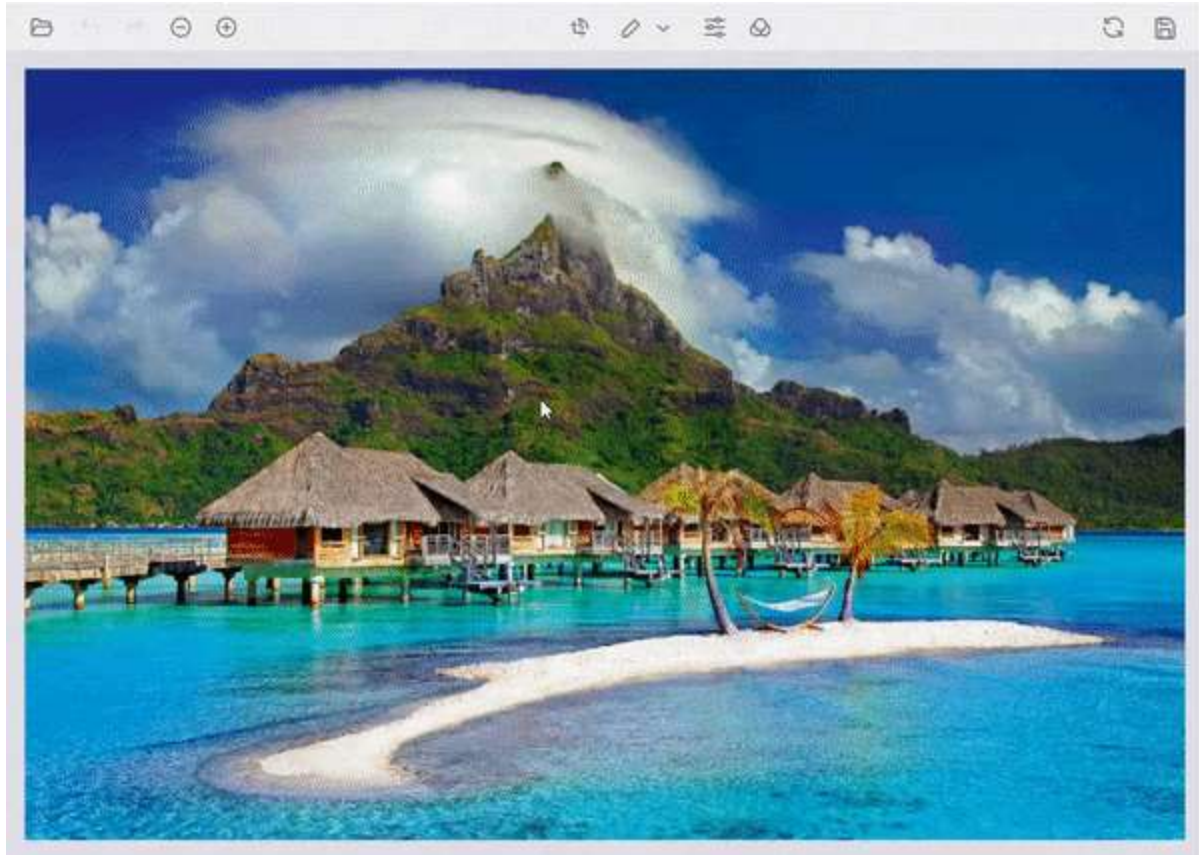
## End-user capabilities in the Image Editor component

The following operations are available for end-users and the same is explained briefly in these sections.

### Open an image

To open an image in the image editor, do the following steps.

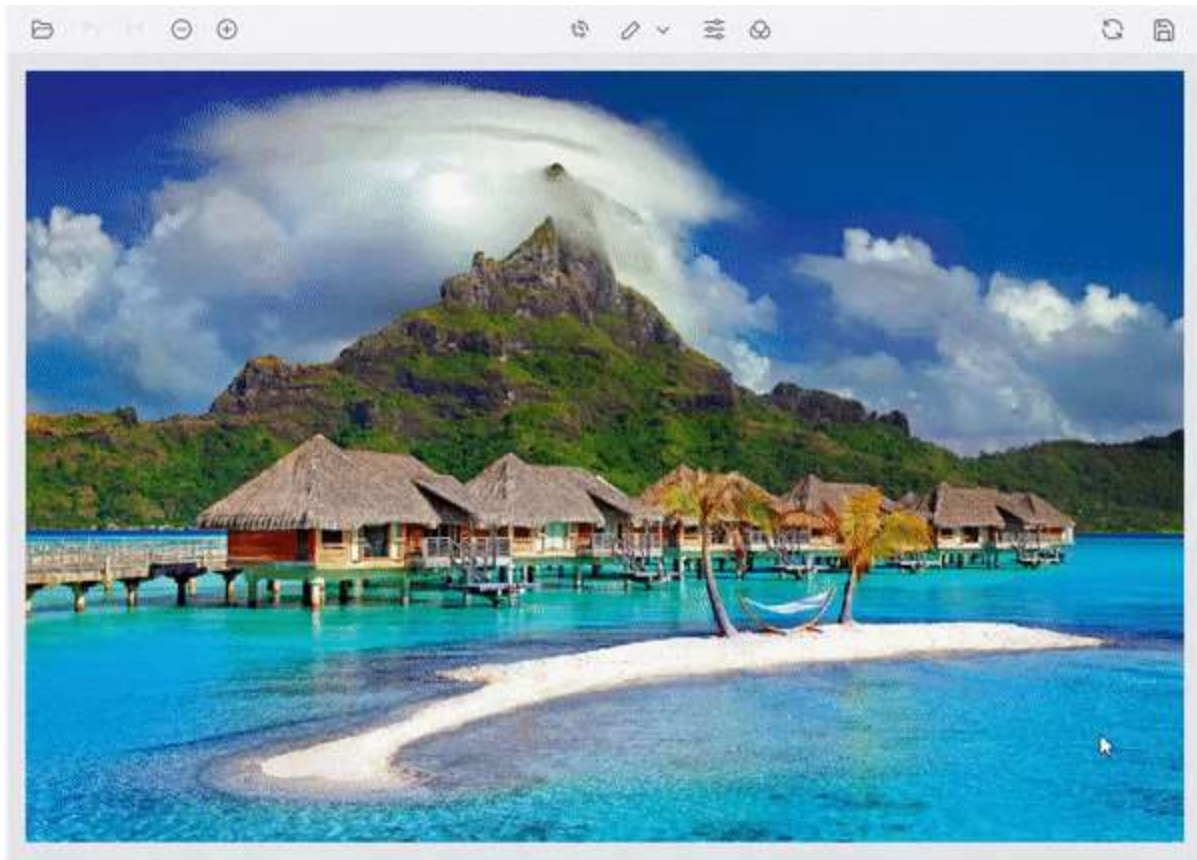
- Click the Open icon from the left side of the toolbar.
- The file explorer lists only JPEG, PNG, JPG format files.
- Select the image from the list of the images from the file explorer window.



### Zooming

Image zooming can be performed in the following ways.

- Using toolbar.
- Using pinch zoom in touch enabled devices.
- Using mouse wheel.
- Using keyboard.



#### *Using toolbar*

To zoom in or out the image in the image editor, do the following steps.

- The Zoom In/ Out option only enabled after opening the image.

#### *Using pinch*

To zoom in or out the image in the image editor, do the following steps.

- Touch with two fingers to perform zooming.
- Zoom in and out controlled by touch gestures.

#### *Using Mouse wheel*

To zoom in or out the image in the image editor, do the following steps.

- Press the ctrl key and scroll the mouse wheel to perform zooming.
- The zoom in and out controlled by the mouse wheel.

#### *Using keyboard*

To zoom in or out the image in the image editor, do the following steps.

- Press the ctrl key with '+' button from the keyboard to zoom in an image.
- Press the ctrl key with '-' button from the keyboard to zoom out an image.



### Panning

To pan an image in the image editor, do the following steps.

- Click on the image and do dragging to move or pan the image.
- Panning option will be enabled in the following two cases.
- If the selection is applied for cropping an image.
- If the image size exceeds the canvas size while zooming an image.



### Cropping and image transformation

To crop an image in the image editor, do the following steps.

- Cropping can be performed based on the selection in an image editor.
- To perform selection, click the crop button in the toolbar which opens the contextual toolbar that shows crop selection options, rotate options, and flip options.
- Click the crop selection button and select the type of selection such as custom, circle, square, and ratio selection from the popup.
- Once selection is completed, do panning to move the image to get the cropped region.
- Utilize the rotate or flip buttons to execute the image transformation, including any inserted annotations.
- Once the cropping region is finalized in the image click the tick icon at the top right of the toolbar to crop the image.



### Annotations

To add annotations to an image in the image editor, do the following steps.

- To add annotation, click the annotation button in the toolbar and select the type of annotations such as Line, Rectangle, Ellipse, Path, Arrow, Text, or Freehand drawing to be inserted to the image editor.
- Once the annotation is added to the image, that can be repositioned by clicking and dragging the annotations using mouse as well as resized by clicking and resizing the selection circle to be placed around the annotations.
- To rotate annotations, you can simply grab the circle located at the bottom of the annotation. The rotation can be applicable to all the annotations except text annotation.
- Customize the annotations by changing their color, stroke width, font family, and font size through the contextual toolbar. The contextual toolbar will be enabled whenever the annotations are selected.
- When annotations are selected in the Image Editor, the quick access toolbar becomes active, providing convenient access to various actions such as duplicating, deleting, or editing text associated with the selected annotation. This toolbar enables users to perform these common operations quickly and efficiently, streamlining their workflow and enhancing the overall editing experience.





### Filtering and fine-tune

To perform fine-tuning on an image in the image editor, do the following steps.

- Click the fine-tune button which displays the list of fine-tuning available in the image editor.
- Click one of the fine-tune options from the list of options which shows a slider to adjust the corresponding filter.
- Click on the canvas or tick icon at the right corner of the toolbar in the image editor to apply the modifications.

To apply filters on an image in the image editor, do the following steps.

- Click the filter button which displays the list of filters available in the image editor.
- Click the filter from list of options to apply the corresponding filter to an image.
- Click on the canvas or tick icon at the right corner of the toolbar in the image editor to apply the modifications.



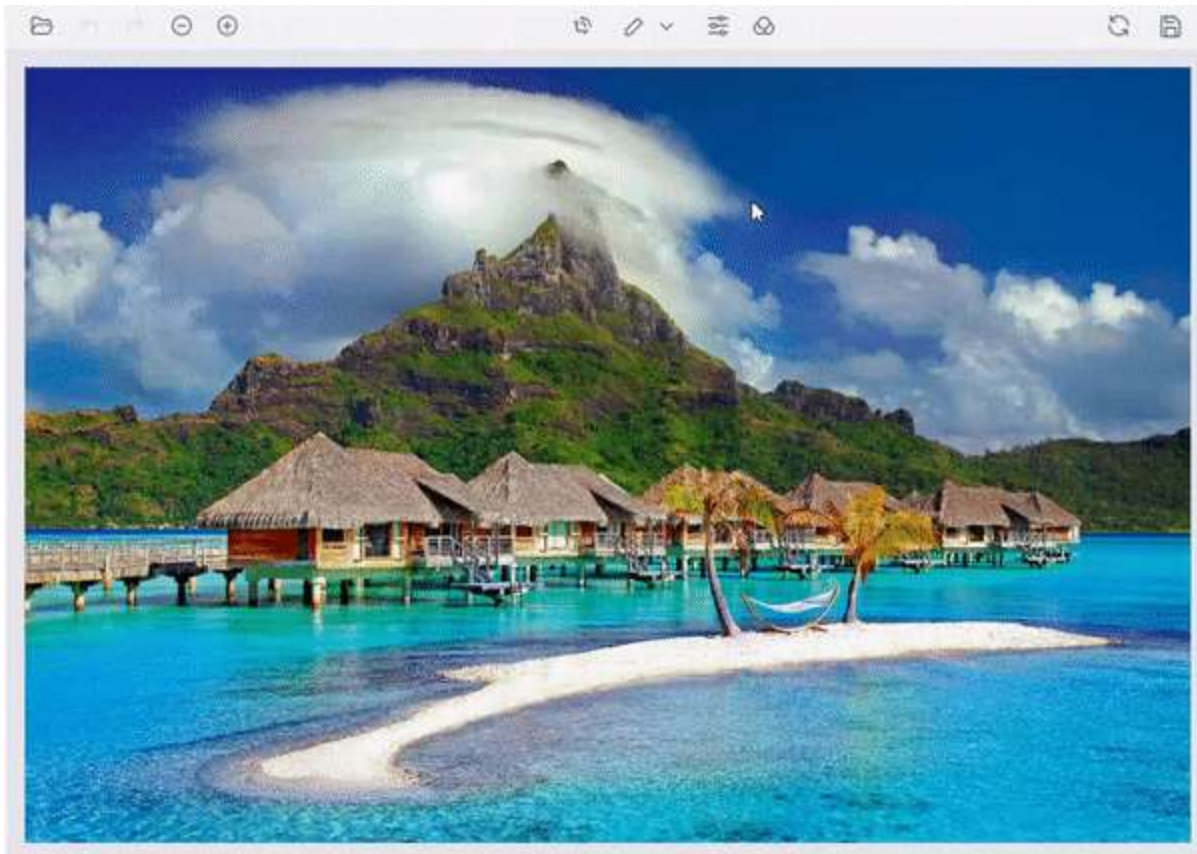
Note:

The Filters and Fine-tunes feature are not accessible within Safari due to compatibility limitations.

#### [Undo and redo the operations](#)

To undo and redo the actions performed in an image editor, do the following steps.

- The undo button will be enabled once the action is performed in an image editor.
- The redo button will be enabled once the undo action is performed in an image editor.
- Click the undo or redo button at the left side of the toolbar to perform undo and redo operation.
- Ctrl + Z and Ctrl + Y facilitates this process by allowing users to undo and redo actions, respectively.



### Reset an image

To revert all the changes done in an image editor, do the following steps.

- Click the reset button which is located on the right side of the toolbar.
- This will revert all the changes performed in the image editor.

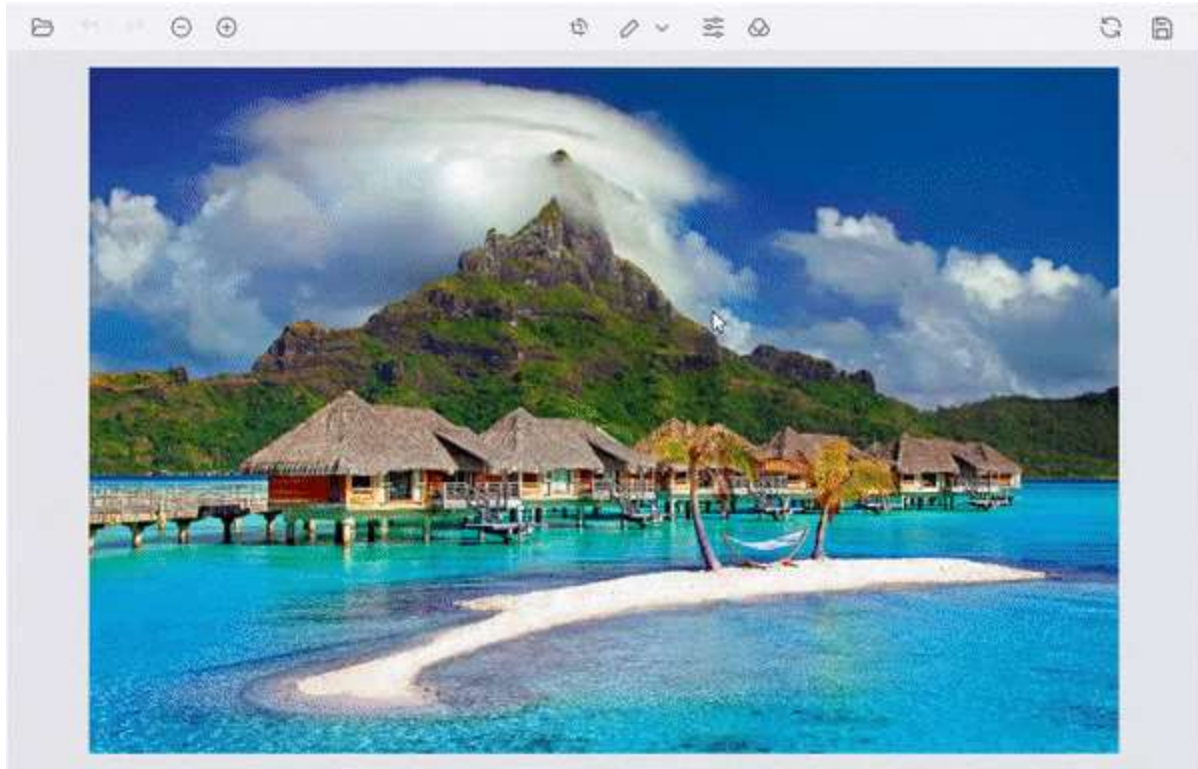




### Export an image

To save the modified image in an image editor, do the following steps.

- Click the save button which is located on the right side of the toolbar.
- Ctrl + S facilitates this process by providing users with the ability to save the image.
- Select the type of file to be saved from the popup to save with current modification done in an image.



### Open and save in the Angular Image Editor component

The Image Editor component supports opening the image by using a hosted/online URL, Image Data, or base64. It also supports save options like image and base64.

#### Open

The Image Editor component opens an image by using base64, Image Data, or a hosted/online URL using the [open](#) method. It also opens an image by clicking the open button from the toolbar. The supported file types are PNG, JPEG, SVG, and base64.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()" >
    [toolbar]="toolbar"></ejs-imageeditor>
  `
})
```

```

        </div>
    </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public toolbar: string[] = [];
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
    }
}

```

### MAIN.TS

```

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

```

### Save

The Image Editor component saves the edited image as Image Data or images like PNG, JPEG, and SVG.

#### Save as ImageData

The [getImageData](#) method is used to get the image as ImageData and this can be loaded to our Image Editor component using the open method.

#### Save as image

The [export](#) method is used to save the modified image as an image, and it accepts a file name and file type as parameters. The file type parameter supports PNG, JPEG, and SVG and the default file type is PNG. It also saves an image by clicking the save button from the toolbar and the supported file types are PNG, JPEG, and SVG.

In the following example, the [export](#) method is used in the button click event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
    imports: [

        ImageEditorModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Image Editor. -->
        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
        </div>
        <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
        </div>`
  })
  export class AppComponent {
    @ViewChild('imageEditor')
    public imageEditorObj?: ImageEditorComponent;
    public toolbar: string[] = [];
    public created(): void {
      if (Browser.isDevice) {
        this.imageEditorObj?.open('./flower.png');
      }
      else {
        this.imageEditorObj?.open('./bridge.png');
      }
    }
    btnClick(): void {
      this.imageEditorObj?.export("PNG", "Syncfusion"); // File type, file
name
    }
  }
}

```

## MAIN.TS

```

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

```

### File opened event

The [fileOpened](#) event is triggered in the Image Editor component after an image is successfully loaded. It provides the [OpenEventArgs](#) as the event argument, which contains two specific arguments:

- **FileName:** This argument is a string that contains the file name of the opened image. It represents the name of the file that was selected or provided when loading the image into the Image Editor.
- **FileType:** This argument is a string that contains the type of the opened image. It specifies the format or file type of the image that was loaded, such as PNG, JPEG, or SVG.

By accessing these arguments within the [fileOpened](#) event handler, you can retrieve information about the loaded image, such as its file name and file type. This can be useful for performing additional actions or implementing logic based on the specific image that was opened in the Image Editor component.

### Saving event

The [saving](#) event is triggered in the Image Editor component when an image is being saved to the local disk. It provides the [SaveEventArgs](#) as the event argument, which includes the following specific arguments:

- **FileName:** This argument is a string that holds the file name of the saved image. It represents the name of the file that will be used when saving the image to the local disk.
- **FileType:** This argument is a string indicating the type or format of the saved image. It specifies the desired file type in which the image will be saved, such as PNG, JPEG, or SVG.
- **Cancel:** This argument is a boolean value that can be set to true in order to cancel the saving action. By default, it is set to false, allowing the saving process to proceed. However, if you want to prevent the saving action from occurring, you can set Cancel to true within the event handler.

By accessing these arguments within the Saving event handler, you can retrieve information about the file name and file type of the image being saved. Additionally, you have the option to cancel the saving action if necessary.

### Created event

The [created](#) event is triggered once the Image Editor component is created. This event serves as a notification that the component has been fully initialized and is ready to be used. It provides a convenient opportunity to render the Image Editor with a predefined set of initial settings, including the image, annotations, and transformations.

### Destroyed event

The [destroyed](#) event is triggered once the Image Editor component is destroyed or removed from the application. This event serves as a notification that the component and its associated resources have been successfully cleaned up and are no longer active.

### Reset an image

The [reset](#) method in the Image Editor component provides the capability to undo all the changes made to an image and revert it back to its original state. This method is particularly useful when multiple adjustments, annotations, or transformations have been applied to an image and you want to start over with the original, unmodified version of the image.

By invoking the [reset](#) method, any modifications or edits made to the image will be undone, and the image will be restored to its initial state. This allows you to easily discard any changes and begin again with the fresh, unaltered image.

### Selection cropping in the Angular Image Editor component

The cropping feature in the Image Editor allows you to select and crop specific regions of an image. It offers different selection options, including custom shapes, squares, circles, and various aspect ratios such as 2:3, 3:2, 3:4, 4:3, 4:5, 5:4, 5:7, 7:5, 9:16, and 16:9.

To perform a selection, you can use the [select](#) method, which allows you to define the desired selection area within the image. Once the selection is made, you can then use the [crop](#) method to crop the image based on the selected region. This enables you to extract and focus on specific parts of the image while discarding the rest.



### Insert custom / square / circle region

The [select](#) method allows to perform selection based on the type of selection. Here, the [select](#) method is used to perform the selection as custom, circle, or square. The selection region can also be customized using the select method based on the parameters below.

type - Specify the type of selection

startX - Specify the x-coordinate of the selection region's starting point

startY - Specify the y-coordinate of the selection region's starting point

width - Specify the width of the selection region

height - Specify the height of the selection region

Here is an example of square selection using the [select](#) method.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  btnClick(): void {
    this.imageEditorObj?.select("Square");
  }
}
```

```
}

```

**MAIN.TS**

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

**Insert selection based on aspect ratio**

The [select](#) method is used to perform the selection with the various aspect ratios such as 2:3, 3:2, 3:4, 4:3, 4:5, 5:4, 5:7, 7:5, 9:16, and 16:9. The selection region can also be customized using the [select](#) method based on the parameters below.

type - Specify the type of selection

startX - Specify the x-coordinate of the selection region's starting point

startY - Specify the y-coordinate of the selection region's starting point

Here is an example of ratio selection using the [select](#) method.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
  }
}
```

```

        else {
            this.imageEditorObj?.open('./bridge.png');
        }
    }
    btnClick(): void {
        this.imageEditorObj?.select("16:9");
    }
}

```

## MAIN.TS

```

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

```

## Resize selections

The selection region can be changed programmatically by using [selectionChanging](#) event. This event is activated during resizing the selection using mouse, and it allows for alterations to the selection region by adjusting the specified properties.

The [SelectionChangeEventArgs](#) is used in these events to customize the selection and it has the following parameters.

[SelectionChangeEventArgs.cction](#) - The type of action such as inserting or resizing

[SelectionChangeEventArgs.cancel](#) - Specifies to cancel the selection.

[SelectionChangeEventArgs.currentSelectionPoint](#) - Represents all the details of the selection including its type, position, width, and height after the current action as [CropSelectionSettings](#).

[SelectionChangeEventArgs.previousSelectionPoint](#) - Represents all the details of the selection including its type, position, width, and height before this current action as [CropSelectionSettings](#)

Here is an example of changing the selection region using the [SelectionChangeEventArgs](#) event.

## Crop an image

The [crop](#) method allows cropping based on the selected region. Here is an example of cropping the selection region using the [crop](#) method.

Here is an example of circle cropping using the [select](#) and [crop](#) method.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
    imports: [

        ImageEditorModule
    ],

```

```

standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  btnClick(): void {
    this.imageEditorObj?.select("Square");
    this.imageEditorObj?.crop();
  }
}

```

## MAIN.TS

```

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

```

## Cropping event

The [cropping](#) event is triggered when performing cropping on the image. This event is passed an object that contains information about the cropping event, such as the start and end point of the selection region. And this event uses [CropEventArgs](#) to handle the cropping action in the image.

The parameter available in the [cropping](#) event is,

CroppingEventArgs.startPoint – The x and y coordinates of a start point as [ImageEditorPoint](#) of the selection region.

CroppingEventArgs.endPoint - The x and y coordinates of an end point as [ImageEditorPoint](#) of the selection region.

CroppingEventArgs.cancel - To cancel the cropping action.

## Annotation in the Angular Image Editor component

The Angular Image Editor allows adding annotations to the image, including text, freehand drawings, and shapes like rectangles, ellipses, arrows, paths, and lines. This gives the flexibility to mark up the

image with notes, sketches, and other visual elements as needed. These annotation tools can help to communicate and share ideas more effectively.

### Text annotation

The text annotation feature in the Image Editor provides the capability to add and customize labels, captions, and other text elements directly onto the image. With this feature, you can easily insert text at specific locations within the image and customize various aspects of the text to meet your requirements.

You have control over the customization options including text content, font family, font style and font size for the text annotation.

### Add a text

The [drawText](#) method in the Angular Image Editor allows you to insert a text annotation into the image with specific customization options. This method accepts the following parameters:

- **x**: Specifies the x-coordinate of the text, determining its horizontal position within the image.
- **y**: Specifies the y-coordinate of the text, determining its vertical position within the image.
- **text**: Specifies the actual text content to be added to the image.
- **fontFamily**: Specifies the font family of the text, allowing you to choose a specific typeface or style for the text.
- **fontSize**: Specifies the font size of the text, determining its relative size within the image.
- **bold**: Specifies whether the text should be displayed in bold style. Set to true for bold text, and false for regular text.
- **italic**: Specifies whether the text should be displayed in italic style. Set to true for italic text, and false for regular text.
- **color**: Specifies the font color of the text, allowing you to define the desired color using appropriate color values or names.
- **isSelected**: Specifies to show the text in the selected state.

By utilizing the [drawText](#) method with these parameters, you can precisely position and customize text annotations within the image. This provides the flexibility to add labels, captions, or other text elements with specific font styles, sizes, and colors, enhancing the visual presentation and clarity of the image.

Here is an example of adding a text in a button click using [drawText](#) method.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
```

```

        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
        </div>
        <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
        </div>`
    ))
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public toolbar: string[] = [];
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            } else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
        btnClick(): void {
            let dimension: any = this.imageEditorObj?.getImageDimension();
            this.imageEditorObj?.drawText(dimension?.x, dimension?.y);
        }
    }
}

```

### MAIN.TS

```

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

```

### Multiline text

The [drawText](#) method in the Angular Image Editor component is commonly used to insert text annotations into an image. If the provided text parameter contains a newline character (\n), the text will be automatically split into multiple lines, with each line appearing on a separate line in the annotation.

Here is an example of adding a multiline text in a button click using [drawText](#) method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, } from '@syncfusion/ej2-angular-image-editor';
@Component({
    imports: [

        ImageEditorModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">

```

```

        <!-- To render Image Editor. -->
        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" ></ejs-imageeditor>
        </div>
        <button class="e-btn e-primary"
(click)="drawClick()">Draw</button>
        <button class="e-btn e-primary"
(click)="btnClick()">Delete</button>
        </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public toolbar: string[] = [];
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
        drawClick(): void {
            this.imageEditorObj?.freeHandDraw(true);
        }
        btnClick(): void {
            this.imageEditorObj?.deleteShape('pen_1');
        }
    }
}

```

## MAIN.TS

```

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

```

### Delete a text

[deleteShape](#) method in the Angular Image Editor allows you to remove a text annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each text annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired text element. By specifying the shapeld associated with the text annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted text annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting a text in a button click using [deleteShape](#) method.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" ></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="drawClick()">Draw</button>
    <button class="e-btn e-primary"
(click)="btnClick()">Delete</button>
    </div>`
  })
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  drawClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawText(dimension?.x,
dimension?.y, 'EnterText');
  }
  btnClick(): void {
    this.imageEditorObj?.deleteShape('shape_1');
  }
}

```

**MAIN.TS**

```

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

```



*Customize font family and text color*

The [shapeChanging](#) event in the Image Editor component is triggered when a text annotation is being modified or changed through the toolbar interaction. This event provides an opportunity to make alterations to the text's color and font family by adjusting the relevant properties.

By leveraging the [shapeChanging](#) event, you can enhance the customization options for text annotations and provide a more tailored and interactive experience within the Image Editor component.

Here is an example of changing the text's color and its font family using the [shapeChanging](#) event.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ShapeChangeEventArgs, ToolbarEventArgs } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
(shapeChanging)="shapeChanging($event)"></ejs-imageeditor>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  public shapeChanging(args: ShapeChangeEventArgs): void {
    if (args.currentShapeSettings?.type === 'Text') {
      args.currentShapeSettings.color = 'red';
      args.currentShapeSettings.fontStyle = ['bold'];
      args.currentShapeSettings.fontSize = 20;
      args.currentShapeSettings.text = 'Syncfusion';
    }
  }
}
```

**MAIN.TS**

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

*Add Additional font family*

The [fontFamily](#) property in the Image Editor control provides the flexibility to incorporate supplementary font families, expanding your options for text styling and ensuring a broader range of fonts can be utilized within your design or content. The font value will be determined by the 'id' property.

By leveraging the [fontFamily](#) property, you can elevate the scope of customization for text annotations, enriching the user experience within the Image Editor control. This enhancement offers a more personalized and dynamic interaction, empowering users to tailor their text styles for a truly engaging editing experience.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[fontFamily]="fontFamily"></ejs-imageeditor>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public fontFamily: any = { default: 'Arial', items: [{id: 'arial', text: 'Arial'}, {id: 'brush script mt', text: 'Brush Script MT'}, {id: 'papyrus', text: 'Papyrus'}, {id: 'times new roman', text: 'Times New Roman'}, {id: 'courier new', text: 'Courier New'}] };
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
}
```

**MAIN.TS**

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

**Freehand drawing**

The Freehand Draw annotation tool in the Angular Image Editor component is a versatile feature that allows users to draw and sketch directly on the image using mouse or touch input. This tool provides a flexible and creative way to add freehand drawings or annotations to the image.

The [freehandDraw](#) method is used to enable or disable the freehand drawing option in the Angular Image Editor component.

Here is an example of using the [freeHandDraw](#) method in a button click event.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
    <button class="e-btn e-primary"
(click)="applyBtnClick()">Apply</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
```

```

    }
  }
  btnClick(): void {
    this.imageEditorObj?.freeHandDraw(true);
  }
  applyBtnClick(): void {
    this.imageEditorObj?.freeHandDraw(false);
  }
}

```

## MAIN.TS

```

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

```

### *Adjust the stroke width and color*

The [shapeChanging](#) event in the Angular Image Editor component is triggered when a freehand annotation is being modified or changed through the toolbar interaction. This event provides an opportunity to make alterations to the freehand annotation's color and stroke width by adjusting the relevant properties.

By leveraging the [shapeChanging](#) event, you can enhance the customization options for freehand annotations and provide a more tailored and interactive experience within the Image Editor component.

Here is an example of changing the freehand draw stroke width and color using the [shapeChanging](#) event.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ShapeChangeEventArgs } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
(shapeChanging)="shapeChanging($event)"></ejs-imageeditor>
    </div>
  </div>`
})
export class AppComponent {

```

```

@ViewChild('imageEditor')
public imageEditorObj?: ImageEditorComponent;
public created(): void {
  if (Browser.isDevice) {
    this.imageEditorObj?.open('./flower.png');
  }
  else {
    this.imageEditorObj?.open('./bridge.png');
  }
}
public shapeChanging(args: ShapeChangeEventArgs): void {
  if (args.currentShapeSettings?.type === 'FreehandDraw') {
    args.currentShapeSettings.strokeColor = 'red',
    args.currentShapeSettings.strokeWidth = 10
  }
}
}

```

### MAIN.TS

```

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

```

#### Delete a freehand drawing

The [deleteShape](#) method in the Angular Image Editor allows you to remove a freehand annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each freehand annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired annotation. By specifying the [shapeld](#) associated with the freehand annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted freehand annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting a freehand annotation in a button click using [deleteShape](#) method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,

```

```

    selector: 'app-root',
    template: `<div class="e-section-control">
      <!-- To render Image Editor. -->
      <div id="wrapperDiv" style="width:550px;height:350px;">
        <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" ></ejs-imageeditor>
      </div>
      <button class="e-btn e-primary"
(click)="drawClick()">Draw</button>
      <button class="e-btn e-primary"
(click)="btnClick()">Delete</button>
    </div>`
  })
  export class AppComponent {
    @ViewChild('imageEditor')
    public imageEditorObj?: ImageEditorComponent;
    public toolbar: string[] = [];
    public created(): void {
      if (Browser.isDevice) {
        this.imageEditorObj?.open('./flower.png');
      }
      else {
        this.imageEditorObj?.open('./bridge.png');
      }
    }
    drawClick(): void {
      this.imageEditorObj?.freeHandDraw(true);
    }
    btnClick(): void {
      this.imageEditorObj?.deleteShape('pen_1');
    }
  }
}

```

## MAIN.TS

```

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

```

## Shape annotation

The Image Editor component provides the ability to add shape annotations to an image. These shape annotations include rectangles, ellipses, arrows, paths, and lines, allowing you to highlight, emphasize, or mark specific areas or elements within the image.

### *Add a rectangle / ellipse / line / arrow / path*

The [drawRectangle](#) method is used to insert a rectangle to the Angular Image Editor component. Rectangle annotations are valuable tools for highlighting, emphasizing, or marking specific areas of an image to draw attention or provide additional context.

The [drawRectangle](#) method in the Angular Image Editor component takes seven parameters to define the properties of the rectangle annotation:

- x: Specifies the x-coordinate of the top-left corner of the rectangle.
- y: Specifies the y-coordinate of the top-left corner of the rectangle.
- width: Specifies the width of the rectangle.
- height: Specifies the height of the rectangle.
- strokeWidth: Specifies the stroke width of the rectangle's border.
- strokeColor: Specifies the stroke color of the rectangle's border.
- fillColor: Specifies the fill color of the rectangle.
- degree: Specifies the degree to rotate the rectangle.
- isSelected: Specifies to show the rectangle in the selected state.

The [drawEllipse](#) method is used to insert a ellipse to the Angular Image Editor component. Ellipse annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawEllipse](#) method in the Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- x: Specifies the x-coordinate of the center of the ellipse.
- y: Specifies the y-coordinate of the center of the ellipse.
- radiusX: Specifies the horizontal radius (radiusX) of the ellipse.
- radiusY: Specifies the vertical radius (radiusY) of the ellipse.
- strokeWidth: Specifies the width of the ellipse's stroke (border).
- strokeColor: Specifies the color of the ellipse's stroke (border).
- fillColor: Specifies the fill color of the ellipse.
- degree: Specifies the degree to rotate the ellipse.
- isSelected: Specifies to show the ellipse in the selected state.

The [drawLine](#) method is used to insert a line to the Angular Image Editor component. Line annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawLine](#) method in the Angular Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- startX - Specifies the x-coordinate of the start point.
- startY - Specifies the y-coordinate of the start point.
- endX - Specifies the x-coordinate of the end point.
- endY - Specifies the y-coordinate of the end point.
- strokeWidth - Specifies the stroke width of the line.
- strokeColor - Specifies the stroke color of the line.
- isSelected: Specifies to show the line in the selected state.

The [drawArrow](#) method is used to insert a arrow to the Angular Image Editor component. Arrow annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawArrow](#) method in the Angular Image Editor component takes seven parameters to define the properties of the ellipse annotation:

- startX - Specifies the x-coordinate of the start point.
- startY - Specifies the y-coordinate of the start point.
- endX - Specifies the x-coordinate of the end point.

- endY - Specifies the y-coordinate of the end point.
- strokeWidth - Specifies the stroke width of the arrow.
- strokeColor - Specifies the stroke color of the arrow.
- arrowStart - Specifies the arrowhead as ImageEditorArrowHeadType at the start of arrow.
- arrowEnd - Specifies the arrowhead as ImageEditorArrowHeadType at the end of the arrow.
- isSelected: Specifies to show the arrow in the selected state.

The [drawPath](#) method is used to insert a path to the Angular Image Editor component. Path annotations are valuable for highlighting, emphasizing, or marking specific areas of an image.

The [drawPath](#) method in the Angular Image Editor component takes three parameters to define the properties of the ellipse annotation:

- points - Specifies collection of x and y coordinates as ImageEditorPoint to draw a path.
- strokeWidth - Specifies the stroke width of the path.
- strokeColor - Specifies the stroke color of the path.
- isSelected: Specifies to show the path in the selected state.

Here is an example of inserting rectangle, ellipse, arrow, path, and line in a button click event.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="rectangleClick()">Rectangle</button>
    <button class="e-btn e-primary"
(click)="ellipseClick()">Ellipse</button>
    <button class="e-btn e-primary"
(click)="lineClick()">Line</button>
    <button class="e-btn e-primary"
(click)="pathClick()">Path</button>
    <button class="e-btn e-primary"
(click)="arrowClick()">Arrow</button>
    </div>`
})
```



```

export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  rectangleClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawRectangle(dimension.x, dimension.y);
  }
  ellipseClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawEllipse(dimension.x, dimension.y);
  }
  lineClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawLine(dimension.x, dimension.y);
  }
  pathClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawPath([
      {x: dimension.x, y: dimension.y},
      {x: dimension.x+50, y: dimension.y+50},
      {x: dimension.x+20, y: dimension.y+50}],
      8);
  }
  arrowClick(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawArrow(
      dimension.x, dimension.y+10,
      dimension.x+50, dimension.y+10, 10);
  }
}

```

## MAIN.TS

```

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

```

## Delete a shape

The [deleteShape](#) method in the Angular Image Editor allows you to remove a shape annotation from the image editor. To use this method, you need to pass the [shapeld](#) of the annotation as a parameter.

The [shapeld](#) is a unique identifier assigned to each shape annotation within the image editor. It serves as a reference to a specific annotation, enabling targeted deletion of the desired annotation. By specifying the [shapeld](#) associated with the shape annotation you want to remove, you can effectively delete it from the image editor.

To retrieve the inserted shape annotations, you can utilize the [getShapeSetting](#) method, which provides a collection of annotations represented by [ShapeSettings](#). This method allows you to access and work with the annotations that have been inserted into the image.

Here is an example of deleting rectangle, ellipse, arrow, path, and line in a button click event.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorCommand, ImageEditorComponent, ShapeChangeEventArgs }
from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" (shapeChanging)="shapeChanging($event)"
showQuickAccessToolbar=false></ejs-imageeditor>
      <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public id?: string | ImageEditorCommand;
  public toolbar: string[] = ['Annotate', 'Line', 'Rectangle', 'Ellipse',
"Circle", "Arrow", "Path"]
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  public shapeChanging(args: ShapeChangeEventArgs): void {
    if (args.action === 'select') {
      this.id = args.currentShapeSettings?.id;
    }
  }
  btnClick(): void {
    if (this.id) {
      this.imageEditorObj?.deleteShape(this.id);
    }
  }
}
```

```
}
}
}
```

## MAIN.TS

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

## Image annotation

The image annotation feature in the Image Editor provides the capability to add and customize images directly onto the image. With this feature, you can easily insert image or icons at specific locations within the image and customize various aspects of the image to meet your requirements. You have control over the customization options including rotate, flip, transparency for the image annotation.

### *Add an image annotation.*

The [drawImage](#) method serves the purpose of inserting an image into the Image Editor control, allowing for image annotations to be added. These image annotations can be used for various purposes, such as adding logos, watermarks, or decorative elements to the image.

The [drawImage](#) method in the Image Editor control takes six parameters to define the properties of the rectangle annotation:

- data: Specified the image data or url of the image to be inserted.
- x: Specifies the x-coordinate of the top-left corner of the image.
- y: Specifies the y-coordinate of the top-left corner of the image.
- width: Specifies the width of the image.
- height: Specifies the height of the image.
- isAspectRatio: Specifies whether the image is rendered with aspect ratio or not.
- degree: Specifies the degree to rotate the image.
- opacity: Specifies the value for the image.
- isSelected: Specifies to show the image in the selected state.

In the following example, you can use the [drawImage](#) method in the button click event.

## APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [
    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
```

```

template: `<div class="e-section-control">
  <!-- To render Image Editor. -->
  <div id="wrapperDiv" style="width:550px;height:350px;">
    <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    <button class="e-btn e-primary" (click)="clickBtn()">Add
Image</button>
  </div>
</div>`
))
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  clickBtn(): void {
    let dimension: any = this.imageEditorObj?.getImageDimension();
    this.imageEditorObj?.drawImage('./flower.png', dimension?.x,
dimension?.y, 100, 100, true, 0);
  }
}

```

### MAIN.TS

```

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

```

### Transform in the Angular Image Editor component

The Image Editor provides a range of transformation options for manipulating both the image and its annotations. These options include rotation, flipping, zooming, and panning. These transformations offer flexibility in adjusting the image and enhancing its visual appearance.

#### Rotate an image

The Image Editor allows to rotate the image and its annotations by a specific number of degrees clockwise or anti-clockwise using [rotate](#) method. This method takes a single parameter: the angle of rotation in degrees. A positive value will rotate the image clockwise, while a negative value will rotate it anti-clockwise.

Here is an example of rotating an image in a button click event.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'

```

```

import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  btnClick(): void {
    this.imageEditorObj?.rotate(90);
  }
}

```

## MAIN.TS

```

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

```

## Flip an image

The Image Editor provides the [flip](#) method, which allows you to flip both the image and its annotations either horizontally or vertically. This method takes a single parameter of type `ImageEditorDirection`, which specifies the direction in which the flip operation should be applied.

The [Direction](#) parameter accepts two values: 'Horizontal' and 'Vertical'. When you choose 'Horizontal', the image and annotations will be flipped along the horizontal axis, resulting in a mirror effect. On the other hand, selecting 'Vertical' will flip them along the vertical axis, producing a vertical mirror effect.

Here is an example of flipping an image in a button click event.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { Direction, ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()" >
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="btnClick()">Click</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  btnClick(): void {
    this.imageEditorObj?.flip(Direction.Horizontal); // Horizontal flip
  }
}

```

**MAIN.TS**

```

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

```

**Straightening in the Angular Image Editor control**

The straightening feature in an Image Editor allows users to adjust an image by rotating it clockwise or counter clockwise. The rotating degree value should be within the range of -45 to +45 degrees for

accurate straightening. Positive values indicate clockwise rotation, while negative values indicate counter clockwise rotation.

#### *Apply straightening to the image*

The Image Editor control includes a [straightenImage](#) method, which allows you to adjust the degree of an image. This method takes one parameter that define how the straightening should be carried out:

- **degree:** Specifies the amount of rotation for straightening the image. Positive values indicate clockwise rotation, while negative values indicate counterclockwise rotation.

Here is an example of straightening the image using the [straightenImage](#) method.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"></ejs-
imageeditor>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
}
```

#### **MAIN.TS**

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

### Zoom in or out an image

The Image Editor allows to magnify an image using the [zoom](#) method. This method allows one to zoom in and out of the image and provides a more detailed view of the image's hidden areas. This method takes two parameters to perform zooming.

- zoomFactor - Specifies a value to controlling the level of magnification applied to the image.
- zoomPoint - Specifies x and y coordinates of a point as ImageEditorPoint on image to perform zooming.

Here is an example of zooming an image in a button click event.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ZoomSettingsModel } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" [zoomSettings]="zoomSettings"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary" (click)="zoomInClick()">Zoom
In</button>
    <button class="e-btn e-primary" (click)="zoomOutClick()">Zoom
Out</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public zoomLevel: number = 1;
  public zoomSettings: ZoomSettingsModel = {maxZoomFactor: 30,
minZoomFactor: 0.1};
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
}
```



```

zoomInClick(): void {
    if(this.zoomLevel < 1) {
        this.zoomLevel += 0.1;
    }
    else {
        this.zoomLevel += 1;
    }
    const value: any = this.imageEditorObj?.zoomSettings.maxZoomFactor;
    if(this.zoomLevel > value) {
        this.zoomLevel = value;
    }
    this.imageEditorObj?.zoom(this.zoomLevel) // zoom in
}
zoomOutClick(): void {
    if(this.zoomLevel <= 1) {
        this.zoomLevel -= 0.1;
    }
    else {
        this.zoomLevel -= 1;
    }
    const value: any = this.imageEditorObj?.zoomSettings.minZoomFactor;
    if(this.zoomLevel < value) {
        this.zoomLevel = value;
    }
    this.imageEditorObj?.zoom(this.zoomLevel) // zoom out
}
}

```

## MAIN.TS

```

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

```

### Maximum and Minimum zoom level

The [maxZoomFactor](#) property is a useful feature in the Image Editor that allows you to define the maximum level of zoom permitted for an image. This property sets a limit on how much the image can be magnified, preventing excessive zooming that may result in a loss of image quality or visibility.

By default, the [minZoomFactor](#) value is set to 10, meaning that the image can be zoomed in up to 10 times its original size. This ensures that the zooming functionality remains within reasonable bounds and maintains the integrity of the image.

The [maxZoomFactor](#) property allows you to specify the minimum level of zoom that is allowed for an image. By setting this property, you can prevent the image from being zoomed out beyond a certain point, ensuring that it remains visible and usable even at the smallest zoom level.

By default, the [maxZoomFactor](#) value is set to 0.1, meaning that the image can be zoomed out up to 10 times its original size.

Here is an example of specifying [minZoomFactor](#) and [maxZoomFactor](#) property in [zoomSettings](#) options in an image editor.

### Panning an image

The Image Editor allows to pan an image when the image exceeds the canvas size or selection range. When zooming in on an image or applying a selection for cropping, it is common for the image to exceed the size of the canvas or exceed the selection range. So, the panning is used to view the entire image, by clicking on the canvas and dragging it in the direction they want to move.

#### Panning event

The [panning](#) event is activated when the user begins dragging the image within the canvas. This event provide an opportunity to perform specific actions, like adjusting the position of an image, in response to the gesture of panning. And these event uses [PanEventArgs](#) to handle the panning action when the user starts dragging the image.

The parameter available in the [PanEventArgs](#) events are,

- `PanEventArgs.startPoint` - The x and y coordinates as `ImageEditorPoint` for the start point.
- `PanEventArgs.endpoint` - The x and y coordinates as `ImageEditorPoint` for the end point.
- `PanEventArgs.cancel` – Specifies the boolean value to cancel the panning action.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ZoomSettingsModel } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="panClick()">Pan</button>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public zoomLevel: number = 1;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
```

```

        this.imageEditorObj?.open('./bridge.png');
    }
}
panClick(): void {
    this.imageEditorObj?.zoom(this.zoomLevel) // zoom in
    this.imageEditorObj?.pan(true);
}
}

```

## MAIN.TS

```

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

```

### Zooming event

The [zooming](#) event is triggered when performing zooming the image. This event can be used to perform certain actions, such as updating the position of the image. This event is passed an object that contains information about the zooming event, such as the amount of zooming performed. And this event uses [ZoomEventArgs](#) to handle the zooming action in the image.

The parameter available in the Zooming event is,

- ZoomEventArgs.zoomPoint - The x and y coordinates as ImageEditorPoint for the zoom point.
- ZoomEventArgs.previousZoomFactor - The previous zoom factor applied in the image editor.
- ZoomEventArgs.currentZoomFactor - The current zoom factor to be applied in the image editor.
- ZoomEventArgs.cancel – Specify a boolean value to cancel the zooming action.
- ZoomEventArgs.zoomTrigger - The type of zooming performed in the image editor.

### Rotating event

The [rotating](#) event is triggered when performing rotating the image. This event is passed an object that contains information about the rotating event, such as the amount of rotation performed. And this event uses [RotateEventArgs](#) to handle the rotating action in the image.

The parameter available in the [rotating](#) event is,

- RotateEventArgs.PreviousDegree: The degree of rotation before the recent rotation action was applied in the Image Editor.
- RotateEventArgs.CurrentDegree: The current degree of rotation after the rotation action has been performed in the Image Editor.
- RotateEventArgs.Cancel – Specifies a boolean value to cancel the rotating action.

### Flipping event

The [flipping](#) event is triggered when performing flipping the image. This event is passed an object that contains information about the flipping event, such as the amount of flip performed. And this event uses [FlipEventArgs](#) to handle the flipping action in the image.

The parameter available in the [flipping](#) event is,

- `FlipEventArgs.Direction` - The flip direction as `ImageEditorDirection` to be applied in the image editor.
- `FlipEventArgs.Cancel` - Specifies a boolean value to cancel the flip action.

### Toolbar in the Angular Image Editor component

The toolbars in the Image Editor are a key component for interacting with and editing images. They provide a range of tools and options that can be customized to suit the needs and preferences. Add or remove items from the toolbar to create a personalized set of tools, or they can even create their own custom toolbar from scratch. This flexibility and customization allow them to create a unique image editing experience that is tailored to their specific needs and workflow.

In the Image Editor, the [toolbar](#) property provides the ability to customize the toolbar by adding or removing items, as well as defining a completely custom toolbar. This feature is valuable for creating a personalized image editing experience that aligns with specific requirements and workflows.

#### Built-in toolbar items

Specifies the toolbar items to perform UI interactions. Refer to the built-in toolbar items for the default value.

- Crop
- Transform
- Annotate
- ZoomIn
- ZoomOut
- Open
- Reset
- Save
- Pan

#### Add a custom toolbar items

The [toolbar](#) property in the Image Editor allows to add or remove toolbar items to include only the tools they frequently use, streamlining the editing process and reducing clutter.

Here is an example of adding custom toolbar items to rotate and flip transformation using [toolbar](#) property.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
import { ClickEventArgs } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
```

```

    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Image Editor. -->
        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="customToolbar"
(toolbarItemClicked)="toolbarItemClicked($event)"></ejs-imageeditor>
        </div>
    </div>`
  })
  export class AppComponent {
    @ViewChild('imageEditor')
    public imageEditorObj?: ImageEditorComponent;
    public customToolbar = ['Annotate', "Line", "Rectangle", "Text",
'ZoomIn', 'ZoomOut', {text: 'Custom'}];
    public created(): void {
      if (Browser.isDevice) {
        this.imageEditorObj?.open('./flower.png');
      }
      else {
        this.imageEditorObj?.open('./bridge.png');
      }
    }
    public toolbarItemClicked(args: ClickEventArgs): void {
      if(args.item.text == "Custom") {
        this.imageEditorObj?.rotate(90);
      }
    }
  }
}

```

## 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 a toolbar

The [toolbar](#) property controls the visibility of the toolbar in the Image Editor. When the [toolbar](#) property is set to an empty list, the toolbar is hidden. Conversely, if the [toolbar](#) property contains a list of items, the toolbar is shown, displaying the specified items. This feature provides flexibility for users to personalize their image editing experience.

Here is an example of hiding the toolbar of the image editor using [toolbar](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({

```

```

imports: [
    ImageEditorModule
],
standalone: true,
selector: 'app-root',
template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
        <ejs-imageeditor #imageEditor (created)="created()" >
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
</div>`
})
export class AppComponent {
    @ViewChild('imageEditor')
    public imageEditorObj?: ImageEditorComponent;
    public toolbar: string[] = [];
    public created(): void {
        if (Browser.isDevice) {
            this.imageEditorObj?.open('./flower.png');
        }
        else {
            this.imageEditorObj?.open('./bridge.png');
        }
    }
}

```

## 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 a toolbar item

The [toolbar](#) property is utilized to control the visibility of toolbar items in the Image Editor. By default, the [toolbar](#) property includes the default toolbar items. If you wish to hide the default toolbar items and specify your own set of required items, you need to explicitly define those items in the [toolbar](#) property. This allows you to customize the toolbar by displaying only the specific items you require, tailoring the editing experience to your preferences.

Here is an example of hiding the cropping and selection toolbar items using [toolbar](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
    imports: [

```

```

        ImageEditorModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Image Editor. -->
        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
        </div>
    </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public toolbar: string[] = ['Annotation' , 'Finetune' , 'Filter' ,
'Confirm' , 'Reset' , 'Save' , 'ZoomIn' , 'ZoomOut'];
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
    }
}

```

## MAIN.TS

```

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

```

## Toolbar template

The [toolbarTemplate](#) property in the Image Editor provides the capability to fully customize the toolbar by supplying a custom template. This feature is valuable when you want to create a distinct and personalized image editing experience that goes beyond the default toolbar or the customizable toolbar options offered by the Image Editor. By defining a custom template for the toolbar, you have complete control over its layout, appearance, and functionality. This empowers you to design a unique and tailored toolbar that aligns perfectly with your specific requirements and desired user experience.

Here is an example of using [toolbarTemplate](#) to render only the button to toggle the freehand draw option.

The toolbar of the Image Editor can be replaced with the user specific UI using the [toolbarTemplate](#) property.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'

```

```

import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()">
        <ng-template #toolbarTemplate>
          <button class="e-btn" (click)="btnClick()">Custom</button>
        </ng-template>
      </ejs-imageeditor>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  btnClick(): void {
    this.imageEditorObj?.freeHandDraw(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 Contextual Toolbar

The [toolbarUpdating](#) event is triggered when inserting or selecting annotations, which opens the contextual toolbar in the Image Editor. Within this event, the [toolbarItems](#) property in the [ToolbarEventArgs](#) is utilized to add or remove contextual toolbar items.

In the following example, the contextual toolbar for rectangle will be rendered with only stroke color by excluding fill color and stroke width using [toolbarUpdating](#) event.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```



```

import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ToolbarEventArgs } from '@syncfusion/ej2-
angular-image-editor';
import { ItemModel } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
(toolbarUpdating)="toolbarUpdating($event)"></ejs-imageeditor>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  public toolbarUpdating(args: ToolbarEventArgs): void {
    if (args.toolbarType === 'pen') {
      args.toolbarItems.forEach((item: ItemModel) => {
        if (item.align === 'Center' && (item.tooltipText === 'Stroke Width'
|| item.tooltipText === 'Remove' || item.type === 'Separator')) {
          item.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));

```

### Toolbar item clicked event

The [toolbarItemClicked](#) event is triggered when a toolbar item is clicked in the Image Editor. This event is particularly useful when you have added custom options to both the main toolbar and contextual toolbar, as it allows you to capture the user's interaction with those custom options. By subscribing to the [toolbarItemClicked](#) event, you can execute specific actions or handle logic based on the toolbar item that was clicked.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ClickEventArgs, ItemModel } from '@syncfusion/ej2-angular-
navigations';
import { ImageEditorCommand, ImageEditorComponent } from '@syncfusion/ej2-
angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" (toolbarItemClicked)="toolbarItemClicked($event)"></ejs-
imageeditor>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] | ItemModel[] = [{text: 'Custom'}];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  public toolbarItemClicked(args: ClickEventArgs): void {
    if (args.item.text === 'Custom') {
      this.imageEditorObj?.rotate(90);
    }
  }
}
```

### MAIN.TS

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

### Toolbar created event

The [toolbarCreated](#) event is triggered after the toolbar is created in the Image Editor. This event can be useful when you need to perform any actions or make modifications to the toolbar once it is fully initialized and ready for interaction. By subscribing to the [toolbarCreated](#) event, you can access the toolbar object and perform tasks such as adding event handlers, customizing the appearance, or configuring additional functionality.

### Add an additional contextual Toolbar item to text shape

The contextual toolbar that appears when inserting annotations in the Image Editor is customizable using the [toolbarUpdating](#) event. This event is triggered when the contextual toolbar is rendered, allowing you to modify its contents. To add additional toolbar items to the contextual toolbar, you can access the [toolbarItems](#) property of the object within the event handler. By adding or removing items from the [toolbarItems](#) property based on the Item property, you can customize the options available in the contextual toolbar according to your needs. This gives you the ability to extend the functionality of the contextual toolbar and provide additional tools and options for working with inserted annotations.

Here is an example of adding the custom toolbar item to the contextual toolbar.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorCommand, ImageEditorComponent, ToolbarEventArgs } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [
    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
(toolbarUpdating)="toolbarUpdating($event)"></ejs-imageeditor>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
  }
}
```

```

        else {
            this.imageEditorObj?.open('./bridge.png');
        }
    }
    public toolbarUpdating(args: ToolbarEventArgs): void {
        if (args.toolbarType === 'text') {
            args.toolbarItems?.push({text: 'custom'})
        }
    }
}

```

## MAIN.TS

```

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

```

## Quick access toolbar in the Angular Image Editor component

The quick access toolbars in the Angular Image Editor play a vital role in facilitating interactions with annotations like Rectangle, Ellipse, Line, Arrow, and Path. These toolbars offer a diverse array of tools and options that can be tailored to match the specific requirements and preferences associated with each annotation type. The toolbar is only displayed when an annotation is selected, ensuring a focused and contextual user experience. Users have the flexibility to add or remove items from the toolbar, allowing them to create a personalized set of tools. Additionally, users can also build a completely custom toolbar from the ground up, providing them with complete control over the available options and functionality.

### Add a custom toolbar item

The quick access toolbar that appears when inserting annotations in the Image Editor is customizable using the [quickAccessToolbarOpen](#) event. This event is triggered when the quick access toolbar is opened, allowing you to modify its contents. To add additional toolbar items to the quick access toolbar, you can access the `toolbarItems` property of the `QuickAccessToolbarEventArgs` within the event handler. By adding or removing items from the `toolbarItems` property based on the item property, you can customize the options available in the quick access toolbar according to your needs. This gives you the ability to extend the functionality of the quick access toolbar and provide additional tools and options for working with inserted annotations.

Here is an example of adding the custom toolbar item to the quick access toolbar.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, QuickAccessToolbarEventArgs,
ShapeChangeEventArgs, ToolbarEventArgs } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

```

```

        ImageEditorModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <!-- To render Image Editor. -->
        <div id="wrapperDiv" style="width:550px;height:350px;">
            <ejs-imageeditor #imageEditor (created)="created()"
(quickAccessToolbarOpen)="quickAccessToolbarOpen($event)"></ejs-imageeditor>
        </div>
    </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
        public quickAccessToolbarOpen(args: QuickAccessToolbarEventArgs): void {
            args.toolbarItems = ['Clone']
        }
    }
}

```

## MAIN.TS

```

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

```

## Undo and redo actions in the Angular Image Editor

The undo and redo functionalities provide a way to reverse and repeat editing actions performed on an image. These features are essential for maintaining control and flexibility during the editing process.

In an image editor, the undo and redo history typically have a limited capacity, and the number of steps that can be stored is 16 steps, meaning that the editor keeps track of the most recent 16 actions performed on the image. Once the history reaches its maximum capacity, any new actions beyond the 16th step will result in the removal of the oldest action from the history.

### Undo the action

The undo action in an image editor allows users to revert the most recent editing action or a series of actions back to their previous state. When the undo command is triggered, the image editor undoes the last applied modification, effectively restoring the image to its state before the action was performed. The undo action is useful for correcting mistakes, removing unwanted changes, or exploring different editing options without permanently altering the image.

### Redo the action

The Redo action in an image editor allows users to reapply previously undone actions or modifications to the image. When the redo command is triggered, the image editor reapplies the last action that was undone, bringing the image back to the state it was in after the action was initially applied. The redo is useful when users want to repeat an action that was previously undone or restore changes that were temporarily reversed.

In the following example, the [undo](#) and [redo](#) method is used in the button click event.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [
    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" allowUndoRedo ></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="undoClick()">Undo</button>
    <button class="e-btn e-primary"
(click)="redoClick()">Redo</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  undoClick(): void {
    this.imageEditorObj?.undo()
  }
  redoClick(): void {
    this.imageEditorObj?.redo()
  }
}
```

**MAIN.TS**

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

**Filters in the Angular Image Editor component**

Filters are pre-defined effects that can be applied to an image to alter its appearance or mood. Image filters can be used to add visual interest or to enhance certain features of the image. Some common types of image filters include cold, warm, chrome, sepia, and invert. This can be done by either using the toolbar or the [applyImageFilter](#) method which takes a single parameter: the filter applied to an image.

**Apply filter effect**

The [applyImageFilter](#) method is utilized to apply filters to an image. By passing the desired filter type as the first parameter of the method, specified as [ImageFilterOption](#) the method applies the corresponding filter to the image. This allows for easy and convenient application of various filters to enhance or modify the image based on the chosen filter type.

- filterOption - Specifies the filter options to the image.

In the toolbar, the default filter can be applied by clicking the Filter option in the toolbar and picking the Default filter.

In the following example, you can use the [applyImageFilter](#) method in the button click event.

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ImageFilterOption } from '@syncfusion/ej2-
angular-image-editor';
@Component({
  imports: [
    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"></ejs-
imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="chromeClick()">Chrome</button>
    <button class="e-btn e-primary"
(click)="coldClick()">Cold</button>
```

```

        <button class="e-btn e-primary"
(click)="warmClick()">Warm</button>
        <button class="e-btn e-primary"
(click)="grayScaleClick()">GrayScale</button>
        <button class="e-btn e-primary"
(click)="sepiaClick()">Sepia</button>
        <button class="e-btn e-primary"
(click)="invertClick()">Invert</button>
    </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
        chromeClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Chrome);
        }
        coldClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Cold);
        }
        warmClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Warm);
        }
        grayScaleClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Grayscale);
        }
        sepiaClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Sepia);
        }
        invertClick(): void {
            this.imageEditorObj?.applyImageFilter(ImageFilterOption.Invert);
        }
    }
}

```

## MAIN.TS

```

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

```

## Image filtering event

The [imageFiltering](#) event is triggered when applying filtering on the image. This event is passed an object that contains information about the filtering event, such as the type of filtering.

The parameter available in the [ImageFilterEventArgs](#) event is,

ImageFilterEventArgs.Filter - The type of filtering as ImageFilterOption to be applied in the image editor.



ImageFilterEventArgs.Cancel – Specifies to cancel the filtering action.

Please note that the Filter and Finetune features were unavailable on iOS due to the non-functioning CanvasContext.filter property on this platform.

### Finetune in Angular Image Editor component

Fine-tuning involves making precise adjustments to the settings of an image filter in order to achieve a specific desired effect. It provides control over the intensity and specific aspects of the filter's impact on the image. For example, fine-tuning allows you to modify parameters like brightness, saturation, or other relevant properties to fine-tune the level or quality of the filter's effect. This level of control enables you to achieve the exact look or outcome you want for your image.

#### Adjust the brightness, contrast, or sharpness

The [finetuneImage](#) method is designed to facilitate fine-tuning operations on an image. It accepts two parameters: the first parameter is [ImageFinetuneOption](#) which determines the type of fine-tuning to be applied (brightness, contrast, or sharpness), and the second parameter represents the fine-tuning value, indicating the degree or intensity of the adjustment. This method allows for convenient adjustment of brightness, contrast, or sharpness by specifying the desired type and corresponding value.

The [finetuneImage](#) method is used to perform brightness, contrast, or sharpness fine-tuning by specifying this type as a first parameter and specifying the fine-tuning value as the second parameter of the method.

- finetuneOption - Specifies the finetune options to be performed in the image.
- value - Specifies the value for finetuning the image.

Here is an example of brightness, contrast, and sharpness fine-tuning using the [finetuneImage](#) method.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent, ImageFinetuneOption, } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" ></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="brightnessClick()">Brightness</button>
```

```

        <button class="e-btn e-primary"
(click)="contrastClick()">Contrast</button>
        </div>`
    })
    export class AppComponent {
        @ViewChild('imageEditor')
        public imageEditorObj?: ImageEditorComponent;
        public toolbar: string[] = [];
        public created(): void {
            if (Browser.isDevice) {
                this.imageEditorObj?.open('./flower.png');
            }
            else {
                this.imageEditorObj?.open('./bridge.png');
            }
        }
        brightnessClick(): void {
            this.imageEditorObj?.finetuneImage(ImageFinetuneOption.Brightness,10);
        }
        contrastClick(): void {
            this.imageEditorObj?.finetuneImage(ImageFinetuneOption.Contrast,10);
        }
    }
}

```

### MAIN.TS

```

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

```

### Adjust the hue, exposure, blur, or opacity

The [finetuneImage](#) method is designed to facilitate fine-tuning operations on an image. It accepts two parameters: the first parameter is [ImageFinetuneOption](#) which determines the type of fine-tuning to be applied (hue, exposure, or blur), and the second parameter represents the fine-tuning value, indicating the degree or intensity of the adjustment. This method allows for convenient adjustment of hue, exposure, or blur by specifying the desired type and corresponding value.

- finetuneOption - Specifies the finetune options to be performed in the image.
- value - Specifies the value for finetuning the image.

Here is an example of hue, exposure, and blur fine-tuning using the [finetuneImage](#) method.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component,ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';

```

```

import { ImageEditorComponent, ImageFinetuneOption, } from '@syncfusion/ej2-
angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar" ></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary"
(click)="hueClick()">Hue</button>
    <button class="e-btn e-primary"
(click)="exposureClick()">Exposure</button>
    <button class="e-btn e-primary"
(click)="blurClick()">Blur</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  hueClick(): void {
    this.imageEditorObj?.finetuneImage(ImageFinetuneOption.Hue,10);
  }
  exposureClick(): void {
    this.imageEditorObj?.finetuneImage(ImageFinetuneOption.Exposure,10);
  }
  blurClick(): void {
    this.imageEditorObj?.finetuneImage(ImageFinetuneOption.Blur,10);
  }
}

```

## MAIN.TS

```

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

```

### Finetune value changing event

The [FinetuneValueChanging](#) event is triggered when performing fine-tuning on the image. This event is passed an object that contains information about the fine-tuning event, such as the type of fine-tuning and the value of fine-tuning performed.

The parameter available in the [FinetuneEventArgs](#) event is,

- `FinetuneEventArgs.finetune` - The type of fine-tuning as [ImageFinetuneOption](#) to be applied in the image editor.
- `FinetuneEventArgs.value` - The fine-tuning value to be applied in the image editor.
- `FinetuneEventArgs.cancel` – Specifies a boolean value to cancel the fine-tuning action.

Please note that the Filter and Finetune features were unavailable on iOS due to the non-functioning `CanvasContext.filter` property on this platform.

### Frames

The frame feature in an Image Editor provides users with the capability to add decorative borders or frames around their images. Frames are a visual design element that can enhance the overall appearance and appeal of an image.

#### Apply frame to the Image

The [drawFrame](#) method is a function designed to enable the application of various frame options to an image. This method simplifies the process of adding decorative frames, such as mat, bevel, line, hook, and inset, to an image by allowing users to specify their desired frame type.

Depending on the frame type selected, users may have additional customization options, such as adjusting the frame's thickness, color, texture, or other attributes. This allows for fine-tuning the appearance of the frame to match the image's theme or the user's preferences.

The [drawFrame](#) method in the Image Editor control takes six parameters to define the properties of the rectangle annotation:

- `frameType` - Specified the image data or url of the image to be inserted.
- `Color` - Specifies the color for the frame.
- `gradientColor` - Specifies the gradient color for the frame.
- `size` - Specifies the size of the frame.
- `inset` - Specifies the inset value for line, hook, and inset type frames.
- `offset` - Specifies the offset value for line and inset type frames.
- `borderRadius` - Specifies the border radius for line type frame.
- `frameLineStyle` - Specifies the frame line style for line type frame.
- `lineCount` - Specifies the line count for the line type frame.

In the following example, you can use the `drawFrame` method in the button click event.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
```

```

import { FrameLineStyle, FrameType, ImageEditorComponent } from
'@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary" (click)="matBtn()">Mat</button>
    <button class="e-btn e-primary"
(click)="bevelBtn()">Bevel</button>
    <button class="e-btn e-primary"
(click)="lineBtn()">Line</button>
    <button class="e-btn e-primary"
(click)="insetBtn()">Inset</button>
    <button class="e-btn e-primary"
(click)="hookBtn()">Hook</button>
    </div>`
  })
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  matBtn(): void {
    this.imageEditorObj?.drawFrame(FrameType.Mat, 'red', 'blue', 20, 20,
20, 20, FrameLineStyle.Solid, 1);
  }
  bevelBtn(): void {
    this.imageEditorObj?.drawFrame(FrameType.Bevel, 'red', 'blue', 20,
20, 20, 20, FrameLineStyle.Solid, 1);
  }
  lineBtn(): void {
    this.imageEditorObj?.drawFrame(FrameType.Line, 'red', 'blue', 20, 20,
20, 20, FrameLineStyle.Solid, 1);
  }
  insetBtn(): void {
    this.imageEditorObj?.drawFrame(FrameType.Inset, 'red', 'blue', 20, 20,
20, 20, FrameLineStyle.Solid, 1);
  }
  hookBtn(): void {
    this.imageEditorObj?.drawFrame(FrameType.Hook, 'red', 'blue', 20, 20,
20, 20, FrameLineStyle.Solid, 1);
  }
}

```

```
}  
}
```

## MAIN.TS

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

### Frame changing event

The [frameChanging](#) event is triggered when applying frame on the image. This event provides information encapsulated within an object, which includes details about the frame applied in an image. This information encompasses:

**Frame Type:** This indicates the specific type of frame being applied, whether it's a mat, bevel, line, or hook.

**Customization Values:** These values contain information about any adjustments or modifications made to the frame. For instance, if the frame can be customized with attributes like color, size, or style, these details are conveyed within the event object.

The parameter available in the [FrameChangeEventArgs](#) is

- [FrameChangeEventArgs.previousFrameSetting](#) - The frame settings including size, color, inset, offset, gradient color which is applied before changing the frame.
- [FrameChangeEventArgs.currentFrameSetting](#) - The frame settings including size, color, inset, offset, gradient color which is going to apply after changing the frame.
- [FrameChangeEventArgs.cancel](#) - Specifies a boolean value to cancel the frame changing action.

### Resize

The resize feature in an Image Editor is a valuable tool that empowers users to modify the size or dimensions of an image to meet their specific requirements. Whether it's for printing, web display, or any other purpose, this feature allows users to tailor images to their desired specifications.

#### Apply resize to the image

The Image Editor control includes a [resize](#) method, which allows you to adjust the size of an image. This method takes three parameters that define how the resizing should be carried out:

- **width:** Specifies the resizing width of the image.
- **height:** Specifies the resizing height of the image.
- **isAspectRatio:** Specifies a boolean value indicating whether the image should maintain its original aspect ratio during resizing. When set to true, the image will be resized while preserving its aspect ratio

Here is an example of resizing the image using the [resize](#) method.

## APP.COMPONENT.TS

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

```

import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
@Component({
  imports: [

    ImageEditorModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
      <ejs-imageeditor #imageEditor (created)="created()"
[toolbar]="toolbar"></ejs-imageeditor>
    </div>
    <button class="e-btn e-primary" (click)="aspectBtn()">Aspect
Ratio</button>
    <button class="e-btn e-primary" (click)="nonaspectBtn()">Non
Aspect Ratio</button>
    </div>`
})
export class AppComponent {
  @ViewChild('imageEditor')
  public imageEditorObj?: ImageEditorComponent;
  public toolbar: string[] = [];
  public created(): void {
    if (Browser.isDevice) {
      this.imageEditorObj?.open('./flower.png');
    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
  aspectBtn(): void {
    this.imageEditorObj?.reset();
    this.imageEditorObj?.resize(300, 400, true);
  }
  nonaspectBtn(): void {
    this.imageEditorObj?.reset();
    this.imageEditorObj?.resize(400, 100, false);
  }
}

```

## MAIN.TS

```

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

```

### Resizing event

The [resizing](#) event is triggered when resizing the image. This event provides information encapsulated within an object, which includes details about the previous and current height and width of an image.

The parameter available in [ResizeEventArgs](#) is,

- [ResizeEventArgs.previousWidth](#) - The width of the image before resizing is performed.
- [ResizeEventArgs.previousHeight](#) - The height of the image before resizing is performed.
- [ResizeEventArgs.width](#) - The width of the image after resizing is performed.
- [ResizeEventArgs.height](#) - The height of the image after resizing is performed.
- [ResizeEventArgs.isAspectRatio](#) - The type of resizing performed such as aspect ratio or non-aspect ratio.
- [ResizeEventArgs.cancel](#) - Specifies a boolean value to cancel the resizing action.

### Localization in the Angular Image Editor component

The [Localization](#) library allows you to localize the default text content of the Image Editor. The Image Editor has static text that can be changed to other cultures (Arabic, Deutsch, French, etc.) by defining the [locale](#) value and translation object.

The following list of properties and its values are used in the Image Editor.

Locale key words	Text
-----	-----
Browse	Browse
Crop	Crop
ZoomIn	Zoom In
ZoomOut	Zoom Out
Transform	Transform
Annotation	Annotation
Text	Add Text
Pen	Pen
Reset	Reset
Save	Save
Select	Select
RotateLeft	Rotate Left
RotateRight	Rotate Right
HorizontalFlip	Horizontal Flip
VerticalFlip	Vertical Flip
OK	OK
Cancel	Cancel



FillColor	Fill Color
StrokeColor	Stroke Color
StrokeWidth	StrokeWidth
FontFamily	Font Family
FontStyle	Font Style
FontSize	Font Size
FontColor	Font Color
Pan	Pan
Move	Move
Custom	Custom
Square	Square
Circle	Circle
Rectangle	Rectangle
Line	Line
Default	Default
Bold	Bold
Italic	Italic
BoldItalic	Bold Italic
XSmall	X-Small
Small	Small
Medium	Medium
Large	Large
XLarge	X-Large
ABC	ABC

#### **APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ImageEditorModule } from '@syncfusion/ej2-angular-image-editor'
import { enableRipple } from '@syncfusion/ej2-base'
import { Component, ViewChild } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { ImageEditorComponent } from '@syncfusion/ej2-angular-image-editor';
import { L10n, setCulture } from '@syncfusion/ej2-base';
setCulture('de-DE');
L10n.load({
  'de-DE': {
    'image-editor': {
      'Browse': 'Durchsuche',
      'Crop': 'Ernte',

```

```

        'ZoomIn': 'Hineinzoomen',
        'ZoomOut': 'Rauszoomen',
        'Transform': 'Verwandeln',
        'Annotation': 'Anmerkung',
        'Text': 'Text hinzufügen',
        'Pen': 'Stift',
        'Reset': 'Zurücksetzen',
        'Save': 'Speichern',
        'Select': 'Auswählen',
        'RotateLeft': 'Nach links drehen',
        'RotateRight': 'Drehe nach rechts',
        'HorizontalFlip': 'Horizontaler Flip',
        'VerticalFlip': 'Vertikaler Flip',
        'OK': 'OK',
        'Cancel': 'Absagen',
        'FillColor': 'Füllfarbe',
        'StrokeColor': 'Strichfarbe',
        'StrokeWidth': 'Strichbreite',
        'FontFamily': 'Schriftfamilie',
        'FontStyle': 'Schriftstil',
        'FontSize': 'Schriftgröße',
        'FontColor': 'Schriftfarbe',
        'Pan': 'Pfanne',
        'Move': 'Bewegen',
        'Custom': 'Brauch',
        'Square': 'Quadrat',
        'Circle': 'Kreis',
        'Rectangle': 'Rechteck',
        'Line': 'Linie',
        'Default': 'Standard',
        'Bold': 'Fett gedruckt',
        'Italic': 'Kursiv',
        'BoldItalic': 'Fett Kursiv',
    }
}
});
@Component({
imports: [

    ImageEditorModule
],
standalone: true,
selector: 'app-root',
template: `<div class="e-section-control">
    <!-- To render Image Editor. -->
    <div id="wrapperDiv" style="width:550px;height:350px;">
        <ejs-imageeditor #imageEditor (created)="created()"
locale="de-DE"></ejs-imageeditor>
    </div>
    </div>`
})
export class AppComponent {
    @ViewChild('imageEditor')
    public imageEditorObj?: ImageEditorComponent;
    public created(): void {
        if (Browser.isDevice) {
            this.imageEditorObj?.open('./flower.png');

```

```

    }
    else {
      this.imageEditorObj?.open('./bridge.png');
    }
  }
}

```

### 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 Image Editor component

The Image Editor component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Image Editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  |

| [Section 508](#) Support |  |

| Screen Reader Support |  |

| Right-To-Left Support |  |

| Color Contrast |  |

| Mobile Device Support |  |

| Keyboard Navigation Support |  |

| [Accessibility Checker](#) Validation |  |

| [Axe-core](#) Accessibility Validation |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

```

}
</style>
<div> - All
features of the component meet the requirement.</div>
<div> - Some features of the component do not meet the requirement.</div>
<div> - The component does not meet the requirement.</div>

```

### Keyboard interaction

The Image Editor component followed the keyboard interaction guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Image Editor component.

| **Press** | **To do this** |

| --- | --- |

| **Ctrl + Z** | **Undo the last user action.** |

| **Ctrl + Y** | **Redo the last user action.** |

| **Ctrl + S** | **To save the Image.** |

| **Ctrl + O** | **To open the Image.** |

| **Delete** | **To delete the shape once the shape got selected through mouse click .** |

### Ensuring accessibility

The Image Editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Image Editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Image Editor component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

## In-place Editor

### Getting started with Angular Inplace editor component

This section explains the steps to create a simple **In-place Editor** component and configure its available functionalities in Angular.

#### Getting Started with Angular CLI

The following section explains the steps required to create a simple `angular-cli` application and how to configure `In-place Editor` component.

### Prerequisites

To get started with Syncfusion Angular UI Components, make sure that you have compatible versions of Angular and TypeScript.

- Angular : 6+
- TypeScript : 2.6+

### Setting up an Angular project

Angular provides an easiest way to setup project using Angular CLI [Angular CLI](#) tool.

Install the CLI application globally in your machine.

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

### Create a new application

```
`javascript
ng new syncfusion-angular-app
`
```

Once you have executed the above command you may ask for following options,

- Would you like to add Angular routing?
- Which stylesheet format would you like to use?

By default it install the CSS style base application. To setup with SCSS, pass --style=SCSS argument on create project.

Example code snippet.

```
`javascript
ng new syncfusion-angular-app --style=SCSS
`
```

Navigate to the created project folder.

```
`javascript
cd syncfusion-angular-app
`
```

### Installing Syncfusion In-place Editor 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-inplace-editor](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-inplace-editor --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-inplace-editor@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-inplace-editor@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-inplace-editor:"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 In-place Editor module*

Once you have successfully installed the 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 the following snippet to import the `InPlaceEditorAllModule` in `app.module.ts` from the `@syncfusion/ej2-angular-inplace-editor`.

```
`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 InPlaceEditorAllModule from ej2-angular-inplace-editor package
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
```

```

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    // Registering EJ2 In-place Editor Module
    InPlaceEditorAllModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
`

```

### Adding In-place Editor component

Add the In-place Editor component snippet in `app.component.ts` as follows.

```

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

@Component({
  selector: 'app-root',
  template: <ejs-inplaceeditor id="element" type="Text" value="Andrew"
    [model]="modelObj"></ejs-inplaceeditor>
})
export class AppComponent {
  public modelObj: Object = { placeholder: 'Enter employee name' };
}
`

```

### 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-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-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-richtexteditor/styles/material.css';
@import '../node_modules/@syncfusion/ej2-calendars/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-inplace-editor/styles/material.css';
`
```

The [Custom Resource Generator \(CRG\)](#) is an online web tool, which can be used to generate the custom script and styles for a set of specific components.

This web tool is useful to combine the required component scripts and styles in a single file.

### Running the application

Run the `ng serve` command in command window, it will serve your application and you can open the browser window. Output will appear as follows.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id='container'>
    <div class="control-group">
      <div class="e-heading">
        <h3> Modify Basic Details </h3>
      </div>
      <table>
        <tr>
          <td>Name</td>
          <td class='left'>
            <ejs-inplaceeditor id="element" mode="Inline"
value="Andrew" [model]="elementModel"></ejs-inplaceeditor>
          </td>
        </tr>
        <tr>
          <td>Date of Birth</td>
          <td class='left'>
            <ejs-inplaceeditor id="dateofbirth" type="Date"
mode="Inline" [value]="dateValue" [model]="dateModel"></ejs-inplaceeditor>

```



```

        </td>
      </tr>
      <tr>
        <td>Gender</td>
        <td class='left'>
          <ejs-inplaceeditor id="gender" type="DropDownList"
mode="Inline" value="Male" [model]="genderModel"></ejs-inplaceeditor>
        </td>
      </tr>
    </table>
  </div>
</div>`
})
export class AppComponent {
  public genderData: string[] = ['Male', 'Female'];
  public dateModel: Object = { showTodayButton: true, placeholder: 'Select
date of birth' };
  public dateValue: Date = new Date('04/12/2018');
  public elementModel: Object = { placeholder: 'Enter your name' };
  public genderModel: Object = { dataSource: this.genderData, placeholder:
'Select gender' };
}

```

### MAIN.TS

```

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

```

### Add the In-place Editor with Textbox

By default, the Essential JS2 TextBox component is rendered in **In-place Editor** with [type](#) property sets as Text.

Modify the template in `src/app/app.component.ts` file to render the `ej2-angular-inplace-editor` component.

```

`javascript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: <ejs-inplaceeditor id="element" type="Text" value="Andrew"
[model]="modelObj"></ejs-inplaceeditor>
})
export class AppComponent {
  public modelObj: Object = { placeholder: 'Enter employee name' };
}
`

```

### Configuring DropDownList

You can render the Essential JS2 DropDownList by changing the [type](#) property as [DropDownList](#) and configure its properties and methods using the [model](#) property.

In the following sample, [type](#) and model values are configured to render the [DropDownList](#) component.

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

@Component({
  selector: 'app-root',
  template: <ejs-inplaceeditor id="element" type="DropDownList" mode="Inline"
[model]="modelObj"></ejs-inplaceeditor>
})
export class AppComponent {
  public genderData : string[] = ['Male', 'Female'];
  public modelObj: Object = { placeholder: 'Select gender', dataSource: this.genderData };
}
```

### Integrate DatePicker

You can render the Essential JS2 [DatePicker](#) by changing the [type](#) property as [Date](#) and also configure its properties and methods using the [model](#) property.

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

@Component({
  selector: 'app-root',
  template: <ejs-inplaceeditor id="element" type="Date" [value]="value"
[model]="modelObj"></ejs-inplaceeditor>
})
export class AppComponent {
  public modelObj: Object = { showTodayButton: true };
  public value: Date = new Date('04/12/2018');
}
```

Once you have configured Textbox, DatePicker and DropDownList you will get following output as shown in below,

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
```

```

import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-
editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id='container'>
    <div class="control-group">
      <div class="e-heading">
        <h3> Modify Basic Details </h3>
      </div>
      <table>
        <tr>
          <td>Name</td>
          <td class='left'>
            <ejs-inplaceeditor id="element" mode="Inline"
value="Andrew" [model]="elementModel"></ejs-inplaceeditor>
          </td>
        </tr>
        <tr>
          <td>Date of Birth</td>
          <td class='left'>
            <ejs-inplaceeditor id="dateofbirth" type="Date"
mode="Inline" [value]="dateValue" [model]="dateModel"></ejs-inplaceeditor>
          </td>
        </tr>
        <tr>
          <td>Gender</td>
          <td class='left'>
            <ejs-inplaceeditor id="gender" type="DropDownList"
mode="Inline" value="Male" [model]="genderModel"></ejs-inplaceeditor>
          </td>
        </tr>
      </table>
    </div>
  </div>
`
})
export class AppComponent {
  public genderData: string[] = ['Male', 'Female'];
  public dateModel: Object = { showTodayButton: true, placeholder: 'Select
date of birth' };
  public dateValue: Date = new Date('04/12/2018');
  public elementModel: Object = { placeholder: 'Enter your name' };
  public genderModel: Object = { dataSource: this.genderData, placeholder:
'Select gender' };
}

```

**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

In In-place Editor, the **value** property supports two-way binding functionality. The following example demonstrates how to achieve two-way binding, by using property binding to bind the value to the first In-place Editor component and **ngModel** to bind the model data to the second In-place Editor. The value of the In-place Editor will get changed, when there is any change in the property value or model value.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { FormsModule } from '@angular/forms'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule, FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div id='container'>
    <div class="control-group">
      <table>
        <tr>
          <td><b>TextBox :</b></td>
          <td><ejs-inplaceeditor id="texteditor1" type="Text" mode="Inline"
[ (value) ]="value" [model]="modelObj"></ejs-inplaceeditor></td>
          <td><ejs-inplaceeditor id="texteditor2" type="Text" mode="Inline"
[ (ngModel) ]="value" [model]="modelObj"></ejs-inplaceeditor></td>
        </tr>
        <tr>
          <td><b>Datepicker :</b></td>
          <td><ejs-inplaceeditor id="dateeditor1" type="Date" mode="Inline"
[ (value) ]="dateValue" [model]="dateModel"></ejs-inplaceeditor></td>
          <td><ejs-inplaceeditor id="dateeditor2" type="Date" mode="Inline"
[ (ngModel) ]="dateValue" [model]="dateModel"></ejs-inplaceeditor></td>
        </tr>
        <tr>
          <td><b>DropDownList :</b></td>
          <td><ejs-inplaceeditor id="dropDowneditor1" type="DropDownList"
mode="Inline" [ (value) ]="dropdownValue" [model]="dropDownmodelObj"></ejs-
inplaceeditor></td>
          <td><ejs-inplaceeditor id="dropDowneditor2" type="DropDownList"
mode="Inline" [ (ngModel) ]="dropdownValue" [model]="dropDownmodelObj"></ejs-
inplaceeditor></td>
        </tr>
      </table>
    </div>
  </div>`
})
export class AppComponent {
```

```

public value: string = 'Andrew';
public modelObj: Object = { placeholder: 'Enter employee name' };
public dateModel: Object = { showTodayButton: true, placeholder: 'Select
date of birth' };
public dateValue: Date = new Date('04/12/2018');
public dropdownValue: string = 'Android';
public frameWorkList : string[] = ['Android', 'JavaScript', 'jQuery',
'TypeScript', 'Angular', 'React', 'Vue', 'Ionic'];
public dropDownmodelObj: Object = { placeholder: 'Select frameWorks',
dataSource: this.frameWorkList };
}

```

## MAIN.TS

```

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

```

### Submitting data to the server (save)

You can submit editor value to the server by configuring the [url](#), [adaptor](#) and [primaryKey](#) property.

Property	Usage
<code>url</code>	Gets the URL for server submit action.
<code>adaptor</code>	Specifies the adaptor type that is used by DataManager to communicate with DataSource.
<code>primaryKey</code>	Defines the unique primary key of editable field which can be used for saving data in the data-base.

The [primaryKey](#) property is mandatory. If it's not set, edited data are not sent to the server.

### Refresh with modified value

The edited data is submitted to the server and you can see the new values getting reflected in the **In-place Editor**.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<div id='container'>
  <div className='control-group' style="text-align: center; margin:
100px auto;">

```

```

        Best Employee of the year: <ejs-inplaceeditor id="element"
type="DropDownList" mode="Inline" value="Andrew Fuller" name='Employee'
[url]="url" primaryKey="Employee" adaptor="UrlAdaptor" [model]="model"
(actionSuccess)='actionSuccess($event)'></ejs-inplaceeditor>
    </div>
    <table style='margin:auto;width:50%'>
    <tr>
        <td style="text-align: left">
            Old Value :
        </td>
        <td id="oldValue" style="text-align: left">
        </td>
    </tr>
    <tr>
        <td style="text-align: left">
            New Value :
        </td>
        <td id="newValue" style="text-align: left">
            Andrew Fuller
        </td>
    </tr>
    </table>
</div>
,
    })
    export class AppComponent {
        public frameworkList: string[] = ['Andrew Fuller', 'Janet Leverling',
        'Laura Callahan', 'TypeScript', 'Margaret Hamilt', 'Michael Suyama', 'Nancy
        Davloio', 'Robert King'];
        public model: Object = { dataSource: this.frameworkList, placeholder:
        'Select employee', popupHeight: '200px' };
        public url: String = "https://ej2services.syncfusion.com/development/web-
        services/api/Editor/UpdateData";
        public actionSuccess(e: any): void {
            let newEle: HTMLElement = document.getElementById('newValue') as
            HTMLElement;
            let oldEle: HTMLElement = document.getElementById('oldValue') as
            HTMLElement;
            oldEle.textContent = newEle.textContent;
            newEle.textContent = e.value;
        }
    }
}

```

## 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

- [Types of rendering the editor](#)

## Controls in Angular Inplace editor component

**In-place Editor** renders various components based on the `type` property and it have built-in and injectable components. To use injectable components, inject the required modules into **In-place Editor**. By default, the `type` property set to `Text` and render the `TextBox`.

The following table explains Injectable components module name and built-in components and their types.

Injectable Components		Built in Components	
----- -----			
<a href="#">AutoComplete</a>	(AutoComplete)	<a href="#">TextBox</a>	(Text)
<a href="#">ComboBox</a>	(ComboBox)	<a href="#">DatePicker</a>	(Date)
<a href="#">MultiSelect</a>	(MultiSelect)	<a href="#">DateTimePicker</a>	(DateTime)
<a href="#">TimePicker</a>	(Time)	<a href="#">DropDownList</a>	(DropDownList)
<a href="#">DateRangePicker</a>	(DateRange)	<a href="#">MaskedTextBox</a>	(Mask)
<a href="#">Slider</a>	(Slider)	<a href="#">NumericTextBox</a>	(Numeric)
<a href="#">Rte</a>	(RTE)		
<a href="#">ColorPicker</a>	(Color)		

In the following sample, built-in and injectable based **In-place Editor** components are rendered.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
    <h3> Built-in Controls </h3>
    <table class="table-section">
      <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
DatePicker </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
          <ejs-inplaceeditor id="date" mode="Inline" type="Date"
[model]="dateModel" [value]='dateValue'></ejs-inplaceeditor>
        </td>
      </tr>
      <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
DateTimePicker </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
```

```

        <ejs-inplaceeditor id="dateTime" mode="Inline"
type="DateTime" [value]='dateTimeValue' [model]="dateModel"></ejs-
inplaceeditor>
        </td>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
DropDownList </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="dropDowns" mode="Inline"
type="DropDownList" value="Android" [model]="dropDownModel"></ejs-
inplaceeditor>
        </td>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
MaskedTextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="masked" mode="Inline" type="Mask"
value="123-345-678" [model]="maskModel"></ejs-inplaceeditor>
        </td>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
NumericTextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="numeric" mode="Inline" type="Numeric"
value=10 [model]="numericModel"></ejs-inplaceeditor>
        </td>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
TextBox </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="textbox" mode="Inline" type="Text"
value="Andrew" [model]="textModel"></ejs-inplaceeditor>
        </td>
    </tr>
</table>
<h3> Injectable Controls </h3>
<table class="table-section">
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
AutoComplete </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="autoComplete" mode="Inline"
type="AutoComplete" value="Android" [model]="dropDownModel"></ejs-
inplaceeditor>
        </td>
    </tr>
    <tr>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
ColorPicker </td>
        <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="color" mode="Inline" type="Color"
value="#81aefd"></ejs-inplaceeditor>
        </td>
    </tr>

```



```

        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                ComboBox </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="comboBox" mode="Inline"
                type="ComboBox" value="Android" [model]="dropDownModel"></ejs-inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                DateRangePicker </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="dateRange" type="DateRange"
                mode="Inline" [value]="dateRangeValue" [model]="dateModel"></ejs-
                inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                MultiSelect </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="multiSelect" mode="Inline"
                type="MultiSelect" value="multiSelectValue" [model]="dropDownModel"></ejs-
                inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                RTE </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="rte" mode="Inline" type="RTE"
                value="<p>Enter your content here</p>" [model]="rteModel"></ejs-
                inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                Slider </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="slider" mode="Inline" type="Slider"
                value=20></ejs-inplaceeditor>
            </td>
        </tr>
        <tr>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
                TimePicker </td>
            <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
                <ejs-inplaceeditor id="time" mode="Inline" type="Time"
                [value]="dateValue" [model]="dateModel"></ejs-inplaceeditor>
            </td>
        </tr>
    </table>
</div>
    })
    export class AppComponent {

```

```

public dateModel: Object = { placeholder: 'Select date' };
public dateValue: Date = new Date('11/23/2018');
public dateTimeValue: Date = new Date('11/23/2018 12:30 PM');
public frameWorkList: string[] = ['Android', 'JavaScript', 'jQuery',
'TypeScript', 'Angular', 'React', 'Vue', 'Ionic'];
public dropDownModel: object = { dataSource: this.frameWorkList,
placeholder: 'Select frameworks' };
public multiSelectModel: object = { dataSource: this.frameWorkList,
placeholder: 'Select framework' };
public multiSelectValue: string[] = ['Android'];
public maskModel: object = { mask: '000-000-000' };
public numericModel: object = { placeholder: 'Enter number' };
public textModel: object = { placeholder: 'Enter some text' };
public dateRangeValue: Date[] = [new Date('11/12/2018'), new
Date('11/15/2018')];
public rteModel: object = { placeholder: 'Enter your content here' };
}

```

### MAIN.TS

```

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

```

### Model configuration

Component properties and events can be customized using the **In-place Editor** [model](#) property.

In the following code, the [type](#) defined as the `Date` and `DatePicker` properties are configured through [model](#) property to customize the [DatePicker](#) component at **In-place Editor**.

`typescript

```

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

@Component({
  selector: 'app-root',
  template: `
<ejs-inplaceeditor id="element" type="Date" [model]="dateModel" [value]='dateValue'></ejs-
inplaceeditor>
`
})

export class AppComponent {
  public dateModel: Object = { showTodayButton: true, placeholder: 'Select Date' };
  public dateValue: Date = new Date('04/12/2018');
}

```

## Configuration in Angular Inplace editor component

### Rendering modes

This section explains the supported rendering modes of the **In-place Editor**. Possible Rendering modes are as follows.

- Popup
- Inline

By default, **Popup** mode will be rendered, when opening an editor.

- For **Popup** mode, editable container displays as like tooltip or popover above the element.
- For **Inline** mode, editable container displays as instead of the element. To render **Inline** mode while opening the editor, specify **mode** as **Inline**.

In the following sample, the **In-place Editor** renders with **Inline** mode. You can dynamically switch into another mode by changing the drop-down item value.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, RenderMode } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    InPlaceEditorAllModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> Mode: </td>
          <td>
            <ejs-dropdownlist #dropDown (change)="onChange($event)"
            id='dropDown' width="auto" [dataSource]='modeData' value='Inline'
            placeholder="Select Mode">
              </ejs-dropdownlist>
            </td>
          </tr>
          <tr>
            <td class="sample-td"> Enter your name: </td>
            <td class="sample-td">
              <ejs-inplaceeditor #element id="element" mode="Inline"
              value="Andrew" [model]="model"></ejs-inplaceeditor>
            </td>
          </tr>
        </table>
      </div>
    `
})
export class AppComponent {
  modeData = [
    { mode: 'Popup' },
    { mode: 'Inline' }
  ];
  model = 'Andrew';
  onChange(event: ChangeEventArgs) {
    this.modeData.forEach((mode) => {
      if (mode.mode === event.value) {
        this.modeData = [mode];
      }
    });
  }
}
```

```

        </table>
      </div>
    })
    export class AppComponent {
      @ViewChild('element') editObj?: InPlaceEditorComponent;
      public model: Object = { placeholder: 'Enter some text' };
      public modeData: string[] = ['Inline', 'Popup'];
      public onChange(e: ChangeEventArgs): void {
        const mode: RenderMode = e.itemData.value as RenderMode;
        (this.editObj as InPlaceEditorComponent).mode = mode;
        this.editObj?.dataBind();
      }
    }
  }
}

```

## MAIN.TS

```

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

```

### Pop-up customization

**In-place Editor** popup mode can be customized by using the [title](#) and [model](#) properties in [popupSettings](#) API.

Popup mode rendered by using the Essential JS 2 Tooltip component, so you can use tooltip properties and events to customize the behavior of popup via the [model](#) property of [popupSettings](#) API.

For more details, refer the tooltip documentation [section](#).

In the following sample, popup [title](#) and [position](#) customized using the [popupSettings](#) property. All possible tooltip position data configured in the drop-down, if we change drop down item, selected value bound to [model](#) property and applied it to [Tooltip](#) component. **Tooltip** has following position options.

- TopLeft
- TopCenter
- TopRight
- BottomLeft
- BottomCenter
- BottomRight
- LeftTop
- LeftCenter
- LeftBottom
- RightTop
- RightCenter
- RightBottom

## APP.COMPONENT.TS

```

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

```

```

import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    InPlaceEditorAllModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> Position: </td>
          <td>
            <ejs-dropdownlist #dropDown (change)="onChange($event)"
            id='dropDown' width="auto" [dataSource]='positionData' value='BottomCenter'
            placeholder="Select a position" popupHeight="150px">
              </ejs-dropdownlist>
            </td>
          </tr>
          <tr>
            <td class="edit-heading sample-td"> Enter your name: </td>
            <td class="sample-td">
              <ejs-inplaceeditor #element id="element" mode="Popup"
              value="Andrew" [model]="model" [popupSettings]="popupSettingsModel"></ejs-inplaceeditor>
            </td>
          </tr>
        </table>
      </div>
    `
})
export class AppComponent {
  @ViewChild('element') editObj?: InPlaceEditorComponent;
  public model: Object = { placeholder: 'Enter some text' };
  public popupSettingsModel: object = { title: 'Enter name', model : {
    position: 'BottomCenter' } };
  public positionData: string[] = ['TopLeft', 'TopCenter', 'TopRight', 'BottomLeft', 'BottomCenter', 'BottomRight', 'LeftTop', 'LeftCenter', 'LeftBottom', 'RightTop', 'RightCenter', 'RightBottom'];
  public onChange(e: ChangeEventArgs): void {
    (this.editObj as any).popupSettings.model.position = e.value;
    this.editObj?.dataBind();
  }
}

```

**MAIN.TS**

```

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

```

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

### Event actions for editing

The event action of the editor that enable in the edit mode based on the `editableOn` property, by default `Click` is assigned, the following options are also supported.

- **Click:** The editor will be opened as single click actions.
- **DbClick:** The editor will be opened as double-click actions and it is not applicable for edit icon.
- **EditIconClick:** Disables the editing of event action of input and allows user to edit only through edit icon.

**In-place Editor** get focus by pressing the `tab` key from previous focusable DOM element and then by pressing `enter` key, the editor will be opened.

In the following sample, when switching drop-down item, the selected value assigned to the `editableOn` property. If you changed to `DbClick`, the editor will open when making a double click on the input.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, EditableType } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    InPlaceEditorAllModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> EditableOn: </td>
          <td>
            <ejs-dropdownlist #dropDown (change)="onChange($event)"
            id='dropDown' width="auto" [dataSource]='editableOnData' value='Click'
            placeholder="Select edit type">
              </ejs-dropdownlist>
            </td>
          </tr>
          <tr>
            <td class="sample-td"> Enter your name: </td>
            <td class="sample-td">
              <ejs-inplaceeditor #element id="element" mode="Inline"
              value="Andrew" [model]="model"></ejs-inplaceeditor>
            </td>
          </tr>
        </table>
      `
})
```

```

        </table>
    </div>
}
})
export class AppComponent {
    @ViewChild('element') editObj?: InPlaceEditorComponent;
    public model: Object = { placeholder: 'Enter some text' };
    public editableOnData: string[] = ['Click', 'DbClick', 'EditIconClick'];
    public onChange(e: ChangeEventArgs): void {
        let editType: EditableType = e.itemData.value as EditableType;
        (this.editObj as InPlaceEditorComponent).editableOn = editType;
        this.editObj?.dataBind();
    }
}
}

```

### MAIN.TS

```

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

```

### Action on focus out

Action to be performed when the user clicks outside the container, that means focusing out of editable content and it can be handled by the [actionOnBlur](#) property, by default **Submit** assigned. It also has the following options.

- **Cancel:** Cancels the editing and resets the old content.
- **Submit:** Submits the edited content to the server.
- **Ignore:** No action is performed with this type and allows to edit multiple editors.

In the following sample, when switching drop-down item, the selected value assigned to the `actionOnBlur` property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns';
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, ActionBlur } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
    imports: [
        InPlaceEditorAllModule, DropDownListAllModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `
        <div id='container'>
            <table class="table-section">

```

```

        <tr>
            <td> ActionOnBlur: </td>
            <td>
                <div id="dropDown"></div>
                <ejs-dropdownlist #dropDown (change)="onChange($event)"
id='dropDown' width="auto" [dataSource]='blurActionData' value='Submit'
placeholder="Select blur action">
                </ejs-dropdownlist>
            </td>
        </tr>
        <tr>
            <td class="sample-td"> Enter your name: </td>
            <td class="sample-td">
                <ejs-inplaceeditor #element id="element" mode="Inline"
value="Andrew" [model]="model"></ejs-inplaceeditor>
            </td>
        </tr>
    </table>
</div>
`
    })
    export class AppComponent {
        @ViewChild('element') editObj?: InPlaceEditorComponent;
        public model: Object = { placeholder: 'Enter some text' };
        public blurActionData: string[] = ['Submit', 'Cancel', 'Ignore'];
        public onChange(e: ChangeEventArgs): void {
            let editType: ActionBlur = e.itemData.value as ActionBlur;
            (this.editObj as InPlaceEditorComponent).actionOnBlur = editType;
            this.editObj?.dataBind();
        }
    }
}

```

## MAIN.TS

```

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

```

## Display modes

By default, **In-place Editor** input element highlighted with a dotted underline. To remove dotted underline from input element, add `data-underline="false"` attribute at **In-place Editor** root element.

The following sample, denotes intractable and normal display modes with different samples.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
import { Component } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
@Component({
    imports: [

```



```

    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <h4>Example of data-underline attribute</h4>
      <table class="table-section">
        <tr>
          <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
            Intractable UI </td>
          <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="default" mode="Inline" value="Andrew"
            [model]="model"></ejs-inplaceeditor>
          </td>
        </tr>
        <tr>
          <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6 control-title">
            Normal UI </td>
          <td class="col-lg-6 col-md-6 col-sm-6 col-xs-6">
            <ejs-inplaceeditor id="inline" data-underline="false"
            mode="Inline" value="Andrew" [model]="model"></ejs-inplaceeditor>
          </td>
        </tr>
      </table>
    </div>
  `
})
export class AppComponent {
  public model: Object = { placeholder: 'Enter some text' };
}

```

## 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

- [Types of rendering the editor](#)
- [Animate the editor during popup mode](#)

## Buttons in Angular Inplace editor component

The **In-place Editor** has an option to save and cancel using buttons. The [saveButton](#) and [cancelButton](#) properties accept the [ButtonModel](#) objects for customizing the save and cancel button properties.

Buttons can be show or hide by sets a Boolean value to the [showButtons](#) property.

Without buttons value will be processed via the following ways.

- **[actionOnBlur](#)**: By clicking out side the editor component get focus out and do action based on this property value.
- **[submitOnEnter](#)**: Pressing **Enter** key it performs the submit action, if this property set to **true**.

In the following sample, the [content](#) and [cssClass](#) properties of **Button** value assigned to the [saveButton](#) and [cancelButton](#) properties to customize its appearance. Also check or uncheck a checkbox buttons render or removed from the editor.

To restrict either save or cancel button rendering into a DOM, simply pass empty object **{}** in the [saveButton](#) or [cancelButton](#) properties.

For more details about buttons, refer this documentation [section](#).

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { CheckBoxModule, ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
@Component({
  imports: [
    InPlaceEditorAllModule, CheckBoxModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> ShowButtons: </td>
          <td>
            <ejs-checkbox label="Show" [checked]="true"
              (change)="onChange($event)"></ejs-checkbox>
          </td>
        </tr>
        <tr>
          <td class="sample-td"> Enter your name: </td>
          <td class="sample-td">
            <ejs-inplaceeditor #element id="element" mode="Inline"
              value="Andrew" [model]="model" [saveButton]="saveButton"
              [cancelButton]="cancelButton"></ejs-inplaceeditor>
          </td>
        </tr>
      </table>
    </div>
  `
})
export class AppComponent {
  @ViewChild('element') editorObj?: InPlaceEditorComponent;
  public model: Object = { placeholder: 'Enter some text' };
```

```

public saveButton: object = { content: 'Ok', cssClass: 'e-outline'};
public cancelButton: object = { content: 'Cancel', cssClass: 'e-outline'};
public onChange(e: ChangeEventArgs): void {
    (this.editorObj as InPlaceEditorComponent).showButtons = e.checked as
boolean;
    this.editorObj?.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

- [In-place editor buttons](#)

### Server actions in Angular Inplace editor component

By passing **In-place Editor** component value to the server, the [primaryKey](#) property value must require, otherwise action not performed for remote data.

If the [URL](#) property value is empty, data passing will handled at local and also the [actionSuccess](#) event will trigger with `null` as argument value.

The following arguments are passed to the server when submit actions perform.

Arguments	Explanations
value	For processing edited value, like DB value updating.
primaryKey	For value mapping to the server, like selecting DB.
name	For field mapping to the server, like DB column field mapping.

Find the following sample server codes for defining models and controller functions to configure processing data.

`C#

```

public class SubmitModel
{
    public string Name { get; set; }
    public string PrimaryKey { get; set; }
    public string Value { get; set; }
}

```

```
`C#
public IEnumerable<SubmitModel> UpdateData([FromBody]SubmitModel value)
{
    // User can process data
    return value;
}
`
```

- Server actions successfully done, the [actionSuccess](#) event will be fired with returned server data.
- If the server is not responding, the [actionFailure](#) event will be fired with data, but value not updated in the Editor.

In the following sample, the `actionSuccess` event will trigger once the value submitted successfully into the server. In this sample, both `actionSuccess` and `actionFailure` were configured and resulted value will be converted to chips.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, ActionEventArgs } from '@syncfusion/ej2-angular-inplace-editor';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <span class="content-title"> Select frameworks: </span>
      <ejs-inplaceeditor #element id="element" data-underline="false"
type="MultiSelect" mode="Inline" [value]="value" [model]="model" name="skill"
[url]="url" primaryKey="FrameWork" adaptor="UrlAdaptor" (created)="created()"
(actionSuccess)="actionSuccess($event)"
(actionFailure)="actionFailure($event)"></ejs-inplaceeditor>
    </div>
  `
})
export class AppComponent {
  @ViewChild('element') editObj?: InPlaceEditorComponent;
  public value: string[] = ['JavaScript', 'jQuery'];
  public url = 'https://ej2services.syncfusion.com/development/web-services/api/Editor/UpdateData';
  public frameWorkList: string[] = ['Android', 'JavaScript', 'jQuery', 'TypeScript', 'Angular', 'React', 'Vue', 'Ionic'];
  public model: object = { mode: 'Box', dataSource: this.frameWorkList, placeholder: 'Select skill' };
}
```

```

chipOnCreate() {
  (this.editObj?.element.querySelector('.e-editable-value') as
Element).innerHTML = this.chipCreation(this.editObj?.value, true);
}
public onSuccess(e: ActionEventArgs): void {
  e.value = this.chipCreation(e.value.split(','), true);
}
public onFailure(e: ActionEventArgs): void {
  e.value = this.chipCreation(e.value.split(','), false);
}
public created(): void {
  this.chipOnCreate();
}
chipCreation(data: any, isSuccess: boolean): string | any {
  let value = '<div class="e-chip-list">';
  [].slice.call(data).forEach((val: string) => {
    value += '<div class="e-chip"> <span class="e-chip-text' +
(isSuccess ? '' : 'e-highlight') + '>' + val + '</span></div>';
  });
  value += '</div>';
  return value;
}
}

```

**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 in Angular Inplace editor component**

The Essential JS 2 components load the data either from local data sources or remote data services using the `dataSource` property and it supports the data type of an array or `DataManager`. Also supports different kind of data services such as OData, OData V4, Web API, and data formats such as XML, JSON, JSONP with the help of `DataManager` adaptors.

**Local Data Binding**

To bind local data to the Essential JS 2 components, you can assign a JavaScript array of object or string to the `dataSource` property. The local data source can also be provided as an instance of the `DataManager`.

**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,

```

```

    selector: 'app-root',
    template: `
      <div id='container'>
        <span class="content-title"> Select customer name: </span>
        <ejs-inplaceeditor id="element" mode="Inline" type="DropDownList"
value="Maria Anders" [model]="model"></ejs-inplaceeditor>
      </div>
    `
  })
  export class AppComponent {
    public gameList: Object[] = [
      { Id: '1', Name: 'Maria Anders' },
      { Id: '2', Name: 'Ana Trujillo' },
      { Id: '3', Name: 'Antonio Moreno' },
      { Id: '4', Name: 'Thomas Hardy' },
      { Id: '5', Name: 'Chiristina Berglund' },
      { Id: '6', Name: 'Hanna Moos' }
    ];
    public fields: object = { text: 'Name' };
    public model: object = { dataSource: this.gameList, fields: this.fields,
placeholder: 'Select a customer' };
  }

```

## 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 bind remote data to the Essential JS 2 component, assign service data as an instance of **DataManager** to the **dataSource** property. To interact with remote data source, provide the endpoint URL.

### OData V4

The OData V4 is an improved version of OData protocols, and the **DataManager** can also retrieve and consume OData V4 services. To fetch data from the server by using **DataManager** with the adaptor property configure as **ODataV4Adaptor**.

In the following sample, **In-place Editor** renders a **DropDownList** component and its **dataSource** property configured for fetching a data from the server by using **ODataV4Adaptor** with **DataManager**.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
import { Component } from '@angular/core';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],

```

```

standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <span class="content-title"> Select customer name: </span>
      <ejs-inplaceeditor id="element" mode="Inline" type="DropDownList"
value="Maria Anders" [model]="model"></ejs-inplaceeditor>
    </div>
  `
})
export class AppComponent {
  public fields: object = { text: 'ContactName', value: 'CustomerID' };
  public dataSource: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public query: Query = new
Query().from('Customers').select(['ContactName', 'CustomerID']).take(6);
  public model: object = { dataSource: this.dataSource, placeholder:
'Select a customer', query: this.query, fields: this.fields };
}

```

### 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

Data can fetch from the server by using [DataManager](#) with the adaptor property configure as [WebApiAdaptor](#).

In the following sample, **In-place Editor** render a **DropDownList** component and its **dataSource** property configured for fetching a data from the server by using **WebApiAdaptor** with **DataManager**.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component, OnInit } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { DataManager, WebApiAdaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>

```

```

        <span class="content-title"> Select customer name: </span>
        <ejs-inplaceeditor id="element" mode="Inline" type="DropDownList"
value="Maria Anders" [model]="model"></ejs-inplaceeditor>
    </div>
    `
  })
  export class AppComponent implements OnInit {
    public fields: object = { text: 'ContactName', value: 'CustomerID' };
    public model?: object;
    ngOnInit(): void {
      new DataManager({
        url:
'https://js.syncfusion.com/demos/ejServices/Wcf/Northwind.svc/Customers/',
        adaptor: new WebApiAdaptor
      }).executeQuery(new Query().take(8)).then((e: any) => {
        this.model = { dataSource: e.result.d, placeholder: 'Select a
customer', fields: this.fields };
      });
    }
  }
}

```

## MAIN.TS

```

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

```

## Integration in Angular Inplace editor component

The **In-place Editor** supports adding HTML5 input components using the [template](#) property. The Template property can be given as either a [string](#) or a [query selector](#).

### As a string

The HTML element tag can be given as a string for the template property. Here, the input is rendered as an HTML template.

`typescript

```
template: "<div><input type='text' id='name'></input></div>"
```

,

### As a ng-template

You can render other components inside **In-place editor** using Angular [ng-template](#). We need to use [ng-template](#) inside the [<ejs-inplaceeditor>](#) tag with [#template](#) attribute, which is mandatory to render that template.

`typescript

```
<ng-template #template>
```

```
<input id="date" value="2018-05-23" type="date">
```

```
</ng-template>
```

,



Template mode, the `value` property not handled by the **In-place Editor** component. So, before sending a value to the server, you need to modify at `actionBegin` event, otherwise, an empty string will pass. In the following template sample, before submitting a data to the server, event argument and `value` property content updated in the `actionBegin` event handler.

### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, ActionBeginEventArgs } from '@syncfusion/ej2-angular-inplace-editor';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<div id='container'>
  <span class="content-title"> Select date: </span>
  <ejs-inplaceeditor #element id="element" mode="Inline" value="2018-05-23" (actionBegin)="actionBegin($event)">
    <ng-template #template>
      <input id="date" value="2018-05-23" type="date">
    </ng-template>
  </ejs-inplaceeditor>
</div>
`
})
export class AppComponent {
  @ViewChild('element') editObj?: InPlaceEditorComponent;
  public actionBegin(e: ActionBeginEventArgs): void {
    const value = (<any>(this.editObj as InPlaceEditorComponent).element.querySelector('#date')).value;
    (this.editObj as InPlaceEditorComponent).value = value;
    (<any>e).value = value;
  }
}
```

### MAIN.TS

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

### Validation in Angular Inplace editor component

**In-place Editor** component supports validation and it can be achieved by adding rules to the `validationRules` property, its child property `key` must be same as `name` property, otherwise validation not performed. Submitting data to the server or calling the `validate` method validation executed.

### Validation Rules

In-place Editor has following validation rules, which are used to perform validation.

| Rules | Description | Example |

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

| **required** | The input element must have any input values | a or 1 or - |

| **email** | The input element must have valid **email** format values | <inplace@syncfusion.com> |

| **url** | The input element must have valid **url** format values | <http://syncfusion.com/> |

| **date** | The input element must have valid **date** format values | 12/25/2019 |

| **dateIso** | The input element must have valid **dateIso** format values | 2019-12-25 |

| **number** | The input element must have valid **number** format values | 1.0 or 1 |

| **maxLength** | Input value must have less than or equal to **maxLength** character length | if **maxLength: 5**, [check] is valid and [checking] is invalid |

| **minLength** | Input value must have less than or equal to **minLength** character length | if **minLength: 5**, [testing] is valid and [test] is invalid |

| **rangeLength** | Input value must have value between **rangeLength** character length | if **rangeLength: [4,5]**, [test] is valid and [key] is invalid |

| **range** | Input value must have value between **range** number | if **range: [4,5]**, [4] is valid and [6] is invalid |

| **max** | Input value must have less than or equal to **max** number | if **max: 3**, [3] is valid and [4] is invalid |

| **min** | Input value must have less than or equal to **min** number | if **min: 4**, [5] is valid and [2] is invalid |

### Style in Angular Inplace editor component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

#### Customizing the In-place Editor text

Use the following CSS to customize the default In-place Editor's text content properties like font-family, font-size, color and border bottom.

`CSS

*/ To change color, font family and font size /*

```
.e-inplaceeditor .e-editable-value-wrapper .e-editable-value {
```

```
border-bottom: 2px dotted green;
```

```
color: red;
```

```
font-size: 12px;
```

```
font-family: Segoe UI
```

```
}
,
```

### Customizing the In-place Editor action buttons

Use the following CSS to customize the default In-place Editor's action buttons.

```
`CSS
```

```
/ To change icon color for save button /
```

```
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn .e-btn-icon.e-icons,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn .e-btn-icon.e-icons{
color: green;
}
```

```
/ To change icon color for cancel button /
```

```
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn .e-btn-icon.e-icons, .e-
inplaceeditor-tip .e-editable-action-buttons .e-btn-cancel.e-icon-btn .e-btn-icon.e-icons {
color: red;
}
```

```
/ To change background color for save button /
```

```
.e-inplaceeditor .e-editable-action-buttons .e-btn-save.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-save.e-icon-btn {
background-color: antiquewhite;
}
```

```
/ To change background color for cancel button /
```

```
.e-inplaceeditor .e-editable-action-buttons .e-btn-cancel.e-icon-btn,
.e-inplaceeditor-tip .e-editable-action-buttons .e-btn-cancel.e-icon-btn {
background-color: antiquewhite;
}
```

```
,
```

### Localization in Angular Inplace editor component

#### Localization

Localization library allows you to localize the default text content of the **In-place Editor** to different cultures using the [locale](#) property. **In-place Editor** following keys will be localize based on culture.

```
| Locale key | en-US (default) |
```

```
|-----|-----|
```

```
| save | Close |
```

```
| cancel | Cancel |
```

| loadingText | Loading... |

| editIcon | Click to edit |

| editAreaClick | Click to edit |

| editAreaDoubleClick | Double click to edit |

To load translation object in an application use `load` function of `L10n` class. In the following sample, `French` culture is set to **In-place Editor** and change the tooltip text.

#### **APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit, ViewChild } from '@angular/core';
import { InPlaceEditorComponent, EditableType } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
import { L10n } from '@syncfusion/ej2-base';
@Component({
  imports: [
    InPlaceEditorAllModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> EditableOn: </td>
          <td>
            <ejs-dropdownlist #dropDown (change)="onChange($event)"
            id='dropDown' width="auto" [dataSource]='editableOnData' value='Click'
            placeholder="Select edit type">
              </ejs-dropdownlist>
            </td>
          </tr>
          <tr>
            <td class="sample-td"> Enter your name: </td>
            <td class="sample-td">
              <ejs-inplaceeditor #element id="element" mode="Inline"
              value="Andrew" locale="fr-BE" [model]="model"></ejs-inplaceeditor>
            </td>
          </tr>
        </table>
      </div>
    `
})
export class AppComponent implements OnInit {
  @ViewChild('element') editObj?: InPlaceEditorComponent;
  public model: object = { placeholder: 'Enter some text' };
  public editableOnData: string[] = ['Click', 'DbClick', 'EditIconClick'];
  public onChange(e: ChangeEventArgs): void {
    let editType: EditableType = e.itemData.value as EditableType;
```

```

    (this.editObj as InPlaceEditorComponent).editableOn = editType;
    this.editObj?.dataBind();
  }
  ngOnInit(): void {
    L10n.load({
      'fr-BE': {
        'inplace-editor': {
          'save': 'enregistrer',
          'cancel': 'Annuler',
          'loadingText': 'Chargement...',
          'editIcon': 'Cliquez pour éditer',
          'editAreaClick': 'Cliquez pour éditer',
          'editAreaDoubleClick': 'Double-cliquez pour éditer'
        }
      }
    });
  }
}

```

### 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

Specifies the direction of the **In-place Editor** component using the enableRtl property. For writing systems that require it like Arabic, Hebrew, etc., the direction can be switched to right-to-left.

It will not change based on the locale property.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <span class="content-title"> Enter your name: </span>
      <ejs-inplaceeditor #element id="element" mode="Inline"
value="Andrew" enableRtl=true></ejs-inplaceeditor>
    </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));
```

**Format**

Formatting is a way of representing the value in different format. You can format the following mentioned components with its `format` property, when it passed through the **In-place Editor** [model](#) property.

- [DatePicker](#)
- [DateRangePicker](#)
- [DateTimePicker](#)
- [NumericTextBox](#)
- [Slider](#)
- [TimePicker](#)

**APP.COMPONENT.TS**

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor';
import { Component } from '@angular/core';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <span class="content-title"> Select date: </span>
      <ejs-inplaceeditor #element id="element" type="Date"
[model]="model" [value]="value"></ejs-inplaceeditor>
    </div>
  `
})
export class AppComponent {
  public model: object = { placeholder: 'Select date', format: 'yyyy-MM-dd' };
  public value: Date = new Date('11/23/2018');
```

**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 Inplace editor component

The Inplace editor component followed the accessibility guidelines and standards, including [ADA](#), [Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Inplace editor component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) |  |

| [Section 508 Support](#) |  |

| [Screen Reader Support](#) |  |

| [Right-To-Left Support](#) |  |

| [Color Contrast](#) |  |

| [Mobile Device Support](#) |  |

| [Keyboard Navigation Support](#) |  |

| [Accessibility Checker Validation](#) |  |

| [Axe-core Accessibility Validation](#) |  |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - All features of the component meet the requirement.</div>

<div> - Some features of the component do not meet the requirement.</div>

<div> - The component does not meet the requirement.</div>

### Keyboard interaction

You can use the following key shortcuts to access the Inplace editor without interruptions:

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| Tab | Helps in focusing the Inplace editor on the page and switching between the consecutive Inplace editor bars. |

| Shift + Tab | Helps in focusing the previous Inplace editor bar element on the Inplace editor. |

### Ensuring accessibility

The Inplace editor component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Inplace editor component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Inplace editor component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

## How To

### Dynamic edit mode in Angular Inplace editor component

At component initial load, if you want to open editor state without interacting **In-place Editor** input element, it can be achieved by configuring the [enableEditMode](#) property to `true`.

In the following sample, editor opened at initial load and when toggling a checkbox, it will remove or open the editor.

#### APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { CheckBoxModule, ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
@Component({
  imports: [
    InPlaceEditorAllModule, CheckBoxModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<div id='container'>
  <table class="table-section">
    <tr>
      <td> EnableEditMode: </td>
      <td>
```



```

        <ejs-checkbox id="enable" label="Enable" [checked]="true"
(change)="onChange($event)"></ejs-checkbox>
      </td>
    </tr>
    <tr>
      <td class="sample-td"> Enter your name: </td>
      <td class="sample-td">
        <ejs-inplaceeditor #element id="element" mode="Inline"
value="Andrew" enableEditMode="true" actionOnBlur="Ignore"
[model]="model"></ejs-inplaceeditor>
      </td>
    </tr>
  </table>
</div>
`
  })
  export class AppComponent {
    @ViewChild('element') editObj?: InPlaceEditorComponent;
    public model: object = { placeholder: 'Enter some text' };
    public onChange(e: ChangeEventArgs): void {
      (this.editObj as InPlaceEditorComponent).enableEditMode = e.checked
    }
  }
}

```

### MAIN.TS

```

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

```

### Disable edit mode in Angular Inplace editor component

The edit mode of **In-place Editor** can be disabled by setting the [disabled](#) property value to `true`. In the following sample, when check or uncheck the checkbox, **In-place Editor** component will disable or enable the edit mode.

### APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { CheckBoxModule, ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-buttons';
@Component({
  imports: [
    InPlaceEditorAllModule, CheckBoxModule, ButtonModule
  ],
  standalone: true,

```

```

    selector: 'app-root',
    template: `
    <div id='container'>
    <table class="table-section">
      <tr>
        <td> Disabled: </td>
        <td>
          <ejs-checkbox id="enable" label="Disable" [checked]="false"
(change)="onChange($event)" "></ejs-checkbox>
        </td>
      </tr>
      <tr>
        <td class="sample-td"> Enter your name: </td>
        <td class="sample-td">
          <ejs-inplaceeditor #element id="element" mode="Inline"
value="Andrew" [model]="model"></ejs-inplaceeditor>
        </td>
      </tr>
    </table>
    </div>
    `
  })
  export class AppComponent {
    @ViewChild('element') editObj?: InPlaceEditorComponent;
    public model: object = { placeholder: 'Enter some text' };
    public onChange(e: ChangeEventArgs): void {
      (this.editObj as InPlaceEditorComponent).disabled = e.checked as
boolean;
      this.editObj?.dataBind();
    }
  }

```

## MAIN.TS

```

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

```

### Custom indication in Angular Inplace editor component

You can add custom indication to unsaved input value by using the [actionSuccess](#) event, when data not submitted to the server.

In this sample, the `actionSuccess` event configured and the `URL` property not included. Then submit button clicked, the current editor value saved into input and data sending to server action prevented due to the `URL` property not configured.

But `actionSuccess` event will trigger the handler function with `null` argument values. In handler function data property `primaryKey` value checked, whether it empty or not. If it is empty custom class, added in the `e-value-wrapper` element to customize its styles.

To send input value to local, set the `URL` property as empty.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'
import { Component } from '@angular/core';
import { InPlaceEditorComponent, ActionEventArgs } from '@syncfusion/ej2-angular-inplace-editor';
import { isNullOrUndefined as isNOU } from '@syncfusion/ej2-base';
@Component({
  imports: [
    InPlaceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
<div id='container'>
  <span class="content-title"> Enter your name: </span>
  <ejs-inplaceeditor #element id="element" mode="Inline" value="Andrew"
[model]="model" (actionSuccess)="actionSuccess($event)"></ejs-inplaceeditor>
</div>
`
})
export class AppComponent {
  public model: object = { placeholder: 'Enter some text' };
  public actionSuccess(e: ActionEventArgs | any): void {
    let pk: string = e.data['PrimaryKey'];
    if (isNOU(pk) || pk === '') {
      (document.querySelector('.e-editable-value') as
Element).classList.add('e-send-error');
    }
  }
}

```

## MAIN.TS

```

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

```

## Custom animation in Angular Inplace editor component

In popup mode, the **In-place Editor** rendered with the Essential JS 2 **Tooltip** component. You can use tooltip properties and events to customize the popup by configure properties into the [model](#) property inside the [popupSettings](#) API.

In the following sample, popup animation can be customized by passing animation effect using the **model** property and the dynamic animation effect changes configured from the Essential JS 2 **DropDownList** component **change** event.

## APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { InPlaceEditorAllModule } from '@syncfusion/ej2-angular-inplace-editor'

```

```

import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { InPlaceEditorComponent } from '@syncfusion/ej2-angular-inplace-editor';
import { ChangeEventArgs } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    InPlaceEditorAllModule, DropDownListAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `
    <div id='container'>
      <table class="table-section">
        <tr>
          <td> Open Animation: </td>
          <td>
            <div id="openDropDown"></div>
            <ejs-dropdownlist #openDropDown (change)="onChange($event)"
id='openDropDown' [dataSource]='openAnimateData' value='ZoomIn'
placeholder="Select a animate type" popupHeight="150px">
              </ejs-dropdownlist>
            </td>
          </tr>
          <tr>
            <td class="sample-td"> Enter your name: </td>
            <td class="sample-td">
              <ejs-inplaceeditor #element id="element" mode="Popup"
value="Andrew" [model]="model" [popupSettings]="popupSettings"></ejs-
inplaceeditor>
            </td>
          </tr>
        </table>
      </div>
    `
})
export class AppComponent {
  @ViewChild('element') editObj?: InPlaceEditorComponent;
  public model: object = { placeholder: 'Enter some text' };
  public popupSettings: object = { model: { animation: { open: { effect:
'ZoomIn', duration: 1000, delay: 0 } } } };
  public openAnimateData: string[] = ['None', 'FadeIn', 'FadeZoomIn',
'ZoomIn'];
  public onChange(e: ChangeEventArgs): void {
    (this.editObj as any).popupSettings.model.animation.open.effect =
e.value;
    this.editObj?.dataBind();
  }
}

```

## MAIN.TS

```

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

```

