

USER GUIDE

Essential Studio

for EJ2 Angular

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

Kanban	26
Getting started with Angular Kanban component.....	26
Setup Angular Environment.....	26
Create an Angular Application	26
Installing Syncfusion Kanban package	26
Adding Kanban Module.....	27
Adding CSS reference.....	27
Add Kanban component	28
Run the application	28
Populating cards.....	29
Enable swimlane	30
Columns in Angular Kanban component	31
Single-key mapping	31
Multi-key mapping	32
Header text	33
Header template	33
Toggle columns	34
Stacked headers.....	36
Cards in Angular Kanban component	37
Drag-and-drop.....	37
Header.....	37
Content	38
Template	38
Selection.....	39
Data binding in Angular Kanban component.....	40
Local data	41
Remote data.....	41
Loading data via ajax.....	51
Observables in Angular Kanban component	52
Observable binding using async pipe.....	53
Data binding	53
Perform CRUD operations.....	55
Swimlane in Angular Kanban component.....	58
Render swimlane row	58
Custom row text.....	59

Template	60
Sorting	61
Drag-and-drop.....	62
Create empty row	63
Calculate cards count.....	64
Enable frozen rows	65
Drag and drop in Angular Kanban component	66
Internal drag and drop	66
External drag and drop	68
Sort in Angular Kanban component.....	76
Index.....	76
DataSource Order	78
Custom	79
Change the direction.....	80
Dialog in Angular Kanban component	81
Default Dialog.....	81
Custom Fields.....	83
Dialog Template	86
Prevent Dialog.....	88
Persisting data in server.....	89
Tooltip in Angular Kanban component	95
Tooltip template	95
Validation in Angular Kanban component	96
Minimum card limit.....	97
Maximum card limit	97
Virtualization in Angular Kanban component.....	98
Virtual scrolling	98
Limitations for virtual scrolling	101
Localization in Angular Kanban component	101
Loading translations.....	101
Right to left (RTL)	103
Dimensions in Angular Kanban component.....	104
Auto height and width	104
Height and width in pixel	105
Height and width in percentage	106

Persistence in Angular Kanban component.....	107
Responsive mode in Angular Kanban component.....	108
Layouts.....	108
Scrolling.....	110
Selection.....	110
Style in Angular Kanban component.....	112
To set fixed position to the Kanban header.....	114
Accessibility in Angular Kanban component.....	115
WAI-ARIA attributes.....	116
Keyboard interaction	116
Ensuring accessibility	117
See also	117
How To	117
Header double click in Angular Kanban component	117
Dynamically change columns in Angular Kanban component.....	118
Filter cards in Angular Kanban component.....	120
Search cards in Angular Kanban component.....	121
Ej1 api migration in Angular Kanban component	122
Columns	122
Cards	125
DataSource.....	127
Common.....	127
Swimlane.....	129
Stacked Headers.....	130
WIP Validation.....	131
Keyboard	131
Toggle Columns.....	132
Dialog Editing	132
Dialog Editing Fields	134
Tooltip	135
Context Menu	135
WorkFlows	136
Filtering	136
Searching.....	137
External Drag And Drop	138

Scrolling.....	138
Card Selection and Hover.....	138
Toolbar	139
Responsive	139
State Persistence.....	139
Right to Left - RTL.....	139
Linear Gauge	140
Getting started with Angular Linear gauge component	140
Dependencies.....	140
Setup Angular Environment.....	140
Create an Angular Application	140
Installing Syncfusion Linear Gauge package	140
Registering LinearGauge Module.....	141
Module Injection.....	143
Add Gauge Title.....	143
Axis Range	144
Setting the value of pointer	146
Dimensions in Angular Linear gauge component	147
Size for Linear Gauge	147
Axis in Angular Linear gauge component	148
Setting the start value and end value of the axis.....	148
Line Customization.....	149
Ticks Customization	150
Labels Customization	152
Orientation.....	156
Inverted Axis	157
Opposed Axis.....	158
Multiple Axes	158
Ranges in Angular Linear gauge component	159
Customizing the range	160
Setting the range color for the labels	161
Multiple ranges	162
Gradient Color.....	163
Pointers in Angular Linear gauge component.....	166
Types of pointer	167

Multiple pointers	173
Pointer animation	174
Gradient Color.....	175
Annotations in Angular Linear gauge component	178
Adding annotation	178
Customization	179
Multiple annotations	181
Animation in Angular Linear gauge component	182
User interaction in Angular Linear gauge component.....	184
Tooltip	184
Pointer Drag	189
Print and export in Angular Linear gauge component.....	189
Print.....	189
Export.....	190
Appearance in Angular Linear gauge component.....	193
Customizing the Linear Gauge area	193
Setting up the Linear Gauge title	194
Customizing the Linear Gauge container	195
Fitting the Linear Gauge to the control.....	198
Accessibility in Angular Linear Gauge component.....	199
WAI-ARIA attributes.....	200
Screen reading in Linear Gauge	200
Ensuring accessibility	200
See also	200
Internationalization in Angular Linear gauge component	200
Numeric Format	201
Events in Angular Linear gauge component	201
animationComplete	201
annotationRender	202
axisLabelRender	203
beforePrint	204
dragEnd	204
dragMove	205
dragStart	206
gaugeMouseDown	207

gaugeMouseLeave	207
gaugeMouseMove	208
gaugeMouseUp	209
load	209
loaded	210
resized	211
tooltipRender	211
valueChange.....	212
Methods in Angular Linear gauge component	213
setPointerValue.....	213
setAnnotationValue	214
refresh.....	215
Ej1 api migration in Angular Linear gauge component.....	216
Linear gauge dimensions	216
Line customizaton	216
Ticks.....	217
Labels	219
Axis	220
Ranges.....	220
Bar Pointer	221
Marker Pointer	223
Annotation	224
Tooltip	226
Appearance of Linear Gauge.....	227
Gauge Container type	227
Events.....	227
ListBox.....	229
Getting started with Angular List box component.....	229
Dependencies.....	229
Setup Angular environment.....	230
Create an Angular application	230
Installing Syncfusion ListBox package	230
Adding ListBox module	231
Adding Syncfusion ListBox component.....	231
Adding CSS reference.....	232

Binding data source	232
Run the application	233
Accessibility in Angular List box component.....	234
WAI-ARIA attributes.....	235
Keyboard interaction	235
Ensuring accessibility	236
See also	236
Data binding in Angular List box component.....	236
Local Data.....	236
Remote Data	239
Drag and drop in Angular List box component	239
Single listbox	240
Multiple listbox	240
Dual list box in Angular List box component	242
Icons and templates in Angular List box component.....	243
Icons	243
Templates.....	244
Selection in Angular List box component	245
Single selection	246
Multiple selection	246
Sorting and grouping in Angular List box component	248
Sorting.....	248
Grouping	249
Style and appearance in Angular List box component	250
Horizontal ListBox	251
How To	252
Add items in Angular List box component.....	252
Enable or disable items in Angular List box component.....	253
Enable scroller in Angular List box component.....	254
Form submit in Angular List box component.....	255
Select items in Angular List box component.....	255
ListView	256
Getting started with Angular Listview component.....	256
Setup Angular Environment.....	256
Create an Angular Application	257

Installing Syncfusion Listview package	257
Registering ListView Module	258
Adding CSS Reference	258
Add Listview component	259
Run the application	259
See Also	260
Schematics in Angular Listview component	260
Getting started	260
Dependency and Module injection using Schematics	261
Component generation using Schematics	261
Data binding in Angular Listview component	262
Bind to local data	262
Bind to remote data	264
Grouping in Angular Listview component	265
Customization	266
Check list in Angular Listview component	266
Checkbox Position	267
Nested list in Angular Listview component	268
Customizing templates in Angular Listview component.....	272
Header template	272
Template	273
Group template.....	276
Virtualization in Angular Listview component.....	277
Getting started	278
Scrolling in Angular Listview component.....	280
Style in Angular Listview component.....	283
Customizing ListView	283
Customizing the list items.....	283
Customizing ListView's header.....	284
Customizing group header of ListView	284
Customizing the hover state of ListView control	284
Customizing selected item of ListView control.....	285
Accessibility in Angular ListView component	285
WAI-ARIA attributes.....	286
Keyboard interaction	286

Ensuring accessibility	287
See also	287
Ej1 api migration in Angular Listview component	287
How To	289
Add and remove list items from listview in Angular Listview component	289
Create dual list from listview in Angular Listview component	291
Customize listview as chat window in Angular Listview component	296
Create mobile contact layout from listview in Angular Listview component.....	299
Customize listview with dynamic tags in Angular Listview component	302
Filter list items in the listview in Angular Listview component	306
Get selected items from listview in Angular Listview component	307
Hide checkbox in listview in Angular Listview component	309
List items count in group header in Angular Listview component	315
Manipulate listview as grid layout in Angular Listview component	316
element .e-list-item {	317
Render listview with hyper link navigation in Angular Listview component.....	321
Drag and drop list items in Angular Listview component	322
Load html content via ajax in Angular Listview component	324
Load list items in child list dynamically in Angular Listview component	325
Display spinner until list items are loaded in Angular Listview component	328
Trace all events in listview in Angular Listview component	330
Use dynamic templates in listview based on device in Angular Listview component.....	332
Integrate pager component with listview in Angular Listview component.....	334
Maps	336
Getting started with Angular Maps component.....	336
Dependencies.....	336
Setup Angular Environment.....	337
Create an Angular Application	337
Installing Syncfusion Maps package.....	337
Registering Maps Module	338
Module Injection.....	340
Render shapes from GeoJSON data	341
Bind data source to map	342
Apply Color Mapping	344
Add Title for Maps.....	345

Enable Legend	346
Add Data Label	347
Enable Tooltip	348
Populate data in Angular Maps component	349
Geometry types.....	349
Shape data	350
Data source	350
Data binding.....	352
Binding complex data source	354
Layers in Angular Maps component	357
Multilayer	357
Sublayer	357
Displaying different layer in the view	359
Rendering custom shapes.....	360
Providers	360
Map provider in Angular Maps component.....	360
Bing maps in Angular Maps component	365
Azure maps in Angular Maps component.....	373
Other maps in Angular Maps component.....	380
Customization in Angular Maps component	390
Setting the size for Maps	390
Maps title	391
Setting theme.....	392
Customizing Maps container	393
Customizing Maps area.....	394
Customizing the shapes	395
Setting color to the shapes from the data source	396
Applying border to individual shapes	397
Projection type.....	398
Color mapping in Angular Maps component.....	399
Types of color mapping.....	399
Multiple colors for a single shape	405
Color for items excluded from color mapping	406
Color mapping for bubbles	407
Data labels in Angular Maps component.....	409

Adding data labels.....	409
Customization	411
Label animation.....	412
Smart labels.....	413
Intersect action	414
Adding data label as a template	415
Polygon shape in Angular Maps component	416
Adding polygon shape.....	416
Tooltip	418
Markers in Angular Maps component	423
Adding marker.....	423
Adding marker template	424
Customization	425
Marker shapes	427
Multiple marker groups	428
Customize marker shapes from data source	429
Repositioning the marker using drag and drop	431
Marker zooming.....	435
Marker clustering.....	436
Customization of marker cluster.....	438
Expanding the marker cluster	439
Tooltip for marker	441
Bubble in Angular Maps component	442
Bubble shapes	443
Customization	444
Setting colors to the bubbles from the data source	446
Setting the range of the bubble size	447
Multiple bubble groups.....	448
Enable tooltip for bubble	450
Legend in Angular Maps component	451
Modes of legend	451
Positioning of the legend	452
Legend for shapes	454
Enable legend for bubbles	466
Enable legend for markers	468

Navigation line in Angular Maps component	470
Customization	470
Enabling the arrows	471
User interactions in Angular Maps component.....	473
Zooming	473
Selection.....	486
Highlight	496
Tooltip	502
Print in Angular Maps component.....	506
Print.....	506
Export.....	507
State persistence in Angular Maps component.....	512
State Persistence.....	512
Accessibility in Angular Maps component.....	513
WAI-ARIA attributes.....	514
Screen reading in Maps.....	514
Keyboard Navigation.....	515
Ensuring accessibility	515
See also	515
Internationalization in Angular Maps component.....	515
Globalization	516
Numeric Format	516
Localization in Angular Maps component	517
Methods in Angular Maps component	519
Methods.....	519
getMinMaxLatitudeLongitude	519
Ej1 api migration in Angular Maps component	521
Size Customization	521
Title and Subtitle Customization	521
Zooming Customization	521
Layer Customization.....	523
Shape Customization	524
Marker Customization	525
Bubble Customization	527
DataLabel Customization	529

Legend Customization.....	530
Highlight And Selection Customization.....	533
Navigation Line Customization	534
Tooltip Customization	535
Annotation Cutomization.....	536
Maps Other Properties Customization	537
Events.....	538
How To	540
Annotation in Angular Maps component	540
Custom path in Angular Maps component	543
Drilldown in Angular Maps component.....	544
Marker type in Angular Maps component.....	546
Multiple layer in Angular Maps component	549
Zooming in Angular Maps component	551
MaskedTextBox.....	552
Getting started with Angular Maskedtextbox component.....	552
Dependencies.....	552
Setup angular environment	552
Create a new application	552
Installing Syncfusion MaskedTextBox Package.....	553
Registering MaskedTextBox module	553
Adding CSS reference.....	554
Adding MaskedTextBox component.....	554
Set the mask.....	555
Running the application.....	555
Two way binding	556
Reactive form	557
See Also	558
Mask configuration in Angular Maskedtextbox component	558
Standard mask elements.....	558
Custom mask elements.....	560
Prompt character	561
Accessibility in Angular Maskedtextbox component.....	562
WAI-ARIA attributes.....	563
Ensuring accessibility	563

See also	563
Style appearance in Angular Maskedtextbox component.....	563
Customizing the appearance of MaskedTextBox wrapper element.....	564
Customizing the MaskedTextBox element on hovering	564
How To	564
Perform custom validation using form validator in Angular Maskedtextbox component	564
Set cursor position while focus on the input textbox in Angular Maskedtextbox component	566
Display numeric keypad when focus on mobile devices in Angular Maskedtextbox component ...	567
Customize the ui appearance of the control in Angular Maskedtextbox component	568
Ej1 api migration in Angular Maskedtextbox component	569
Common.....	569
Mask Configuration.....	571
Validation	573
Mention.....	574
Getting started with Angular Mention component.....	574
Dependencies.....	574
Setup Angular environment.....	575
Create an Angular application	575
Installing Syncfusion Mention package.....	575
Adding Mention module	576
Adding CSS reference.....	576
Adding Mention component.....	577
Binding data source	577
Running the application.....	578
Display Mention character.....	579
Working with data in Angular Mention component.....	580
Binding local data.....	580
Binding remote data	583
See Also	584
Filtering data in Angular Mention component	584
Limit the minimum filter character.....	585
Change the filter type	585
Allow spacing between search.....	586
Customize the suggestion item count	587
See Also	588

Sorting in Angular Mention component	588
Template in Angular Mention component	589
Item template	589
Display template	590
No records template	591
Spinner template	592
Localization in Angular Mention component	593
Loading translations.....	593
See Also	594
Customization in Angular Mention component	594
Show or hide mention character	594
Adding the suffix character after selection.....	595
Configure the popup list	596
Trigger character.....	597
Accessibility in Angular Mention component	597
WAI-ARIA attributes.....	598
Keyboard interaction	598
Ensuring accessibility	599
See also	599
Menu Bar	599
Getting started with Angular Menu component	599
Dependencies.....	599
Setup Angular environment.....	600
Create an Angular application	600
Installing Syncfusion Menu Package	600
Adding Menu module	601
Adding Syncfusion Menu component.....	601
Adding CSS reference.....	603
Running the application	603
Group Menu items with separator	604
Icons and sub menu items in Angular Menu component.....	606
Icons	606
Navigation	607
Multilevel nesting	609
See Also	611

Data source binding and custom menu items in Angular Menu component.....	611
Data binding.....	611
Custom menu items.....	614
See Also.....	616
Use case scenarios in Angular Menu component.....	616
Scrollable menu	616
Menu in toolbar	620
Hamburger menu.....	621
Mobile view.....	624
Accessibility in Angular Menu component	627
WAI-ARIA attributes.....	628
Keyboard interaction	629
Ensuring accessibility	629
See also	629
Style and appearance in Angular Menu component	629
How To	630
Change orientation in Angular Menu component.....	630
Customize menu using events in Angular Menu component.....	631
Customize menu items in Angular Menu component.....	633
Create mnemonic ui in menuitem in Angular Menu component.....	638
Change sub menu position in Angular Menu component.....	639
Rounded corner in Angular Menu component.....	641
Change animation settings in Angular Menu component.....	642
Menu item click in Angular Menu component	644
Right to left in Angular Menu component.....	645
Set title icon in Angular Menu component.....	646
Ej1 api migration in Angular Menu component.....	647
Properties.....	647
Methods.....	649
Events.....	651
Message	652
Getting started with Angular Message component.....	652
Setup Angular Environment.....	652
Create an Angular Application	653
Installing Syncfusion Notification package	653

Adding Message module.....	654
Adding CSS Reference	654
Add Message component	654
Run the application	655
Severities in Angular Message component.....	655
Variants in Angular Message component	656
Icons in Angular Message component.....	658
No Icon	658
Custom Icon	658
Close Icon	659
Customization in Angular Message component.....	661
Content Alignment.....	661
Rounded and Square.....	662
CSS Message.....	663
Template in Angular Message component	665
Accessibility in Angular Message component.....	666
WAI-ARIA attributes.....	667
Keyboard interaction	667
Ensuring accessibility	667
See also	667
MultiSelect	667
Getting started with Angular Multi select component.....	667
Dependencies.....	668
Setup angular environment	668
Create a new application	668
Installing Syncfusion MultiSelect package	669
Registering MultiSelect module	669
Adding CSS reference.....	670
Adding MultiSelect component	670
Binding data source	671
Running the application	671
Configure the popup list	672
Data binding in Angular Multi select component.....	673
Binding local data.....	673
Binding remote data	676

Data binding using Async pipe	677
See Also	679
Value binding in ##Platform_Name## Multi select control	679
Primitive Data Types	679
Object Data Types	680
Templates in Angular Multi select component.....	682
Item template	682
Value template.....	683
Group template.....	684
Header template	685
Footer template	687
No records template	688
Action failure template	688
See Also	689
Grouping in Angular Multi select component	690
Customization	691
Grouping with CheckBox.....	691
See Also	692
Filtering in Angular Multi select component	692
Limit the minimum filter character.....	693
Change the filter type	695
Case sensitive filtering	696
Diacritics Filtering.....	697
See Also	698
Custom value in Angular Multi select component.....	698
Virtualization in MultiSelect Dropdown	699
Binding local data.....	700
Binding remote data	701
Customizing items count in virtualization.....	702
Grouping with virtualization	703
Filtering with virtualization	705
Checkbox with virtualization.....	706
Custom value with virtualization	707
Preselect values with virtualization	708
Checkbox in Angular Multi select component.....	710

Select All.....	711
Selection Limit.....	712
Selection Reordering.....	713
See Also	714
Chip customization in Angular Multi select component.....	714
Localization in Angular Multi select component	715
Loading translations.....	716
See Also	717
Style in Angular Multi select component.....	717
Customizing the background color of wrapper element	717
Customizing the appearance of the delimiter wrapper element	717
Customizing the appearance of chips	718
Customizing the dropdown icon's color	718
Customizing the focus color.....	718
Customizing the disabled component's text color	718
Customizing the color of the placeholder text	719
Customizing the placeholder to add mandatory indicator(*).....	719
Customizing the float label element's focusing color	719
Customizing the outline theme's focus color	720
Customizing the background color of focus, hover, and active item's	720
Customizing the appearance of pop-up element	720
Customizing the color of the checkbox.....	720
Accessibility in Angular Multi select component.....	721
WAI-ARIA attributes.....	722
Keyboard interaction	722
Ensuring accessibility	724
See also	724
Form support in Angular Multi select component	724
Template-Driven Forms	724
Reactive Forms.....	725
How To	726
Icons support in Angular Multi select component.....	726
Configure the cascading multi select in Angular Multi select component	727
NumericTextBox.....	730
Getting started with Angular Numerictextbox component.....	730

Dependencies.....	730
Setup angular environment	730
Create a new application	730
Installing Syncfusion NumericTextBox Package.....	731
Registering NumericTextBox module	732
Adding CSS reference.....	732
Adding NumericTextBox component.....	733
Running the application.....	733
Range validation.....	734
Formatting the value.....	735
Precision of numbers	735
Two way binding	736
Reactive form	737
See Also	738
Formats in Angular Numerictextbox component	738
Standard formats	739
Custom formats	739
Globalization in Angular Numerictextbox component	740
Localization	740
Internationalization.....	741
Right to Left(RTL)	743
Accessibility in Angular Numerictextbox component.....	744
WAI-ARIA attributes.....	745
Keyboard interaction	746
Ensuring accessibility	746
See also	747
Style appearance in Angular Numerictextbox component.....	747
Customizing the appearance of NumericTextBox wrapper element.....	747
Customizing the NumericTextBox icons	747
How To	747
Customize the ui appearance of the control in Angular Numerictextbox component	747
Customize the spin buttons up and down arrow in Angular Numerictextbox component.....	748
Customize the step value and hide spin buttons in Angular Numerictextbox component.....	749
Maintain trailing zeros in numerictextbox in Angular Numerictextbox component.....	750
Perform custom validation using form validator in Angular Numerictextbox component.....	751

Prevent nullable input in numerictextbox in Angular Numerictextbox component	752
Ej1 api migration in Angular Numerictextbox component	753
Common.....	753
Globalization	755
Group	755
Numeric configuration	755
Number Formats	757
Validation	757
Pager	758
Getting started with Angular Pager component.....	758
Setup Angular Environment.....	758
Create an Angular Application	758
Installing Syncfusion Pager package	758
Registering Pager Module.....	759
Adding CSS reference.....	760
Adding Pager component	760
Page Size	760
Page Count	761
Run the application	762
Style and appearance in Angular Pager component.....	762
Customizing the Pager	762
Accessibility in Angular Pager component.....	764
WAI-ARIA.....	764
Keyboard navigation	764
PDF Viewer.....	764
Getting Started.....	764
Getting started with Standalone PDF Viewer component.....	764
Getting started with PDF Viewer component.....	769
Feature modules	774
See also	775
Open PDF files in Angular PDF Viewer component	775
Opening a PDF from URL.....	775
Opening a PDF from base64 data	777
Saving PDF file.....	778
Save PDF file to Server	778

Download PDF file as a copy	780
Built-in toolbar in Angular PDF Viewer component.....	780
Show/Hide the default toolbar	781
Show/Hide the default toolbaritem.....	782
Show/Hide the left toolbar with the thumbnails and bookmarks.....	784
See also	785
Navigation in Angular PDF Viewer component.....	785
Toolbar page navigation option	785
Bookmark navigation	787
Thumbnail navigation	788
Hyperlink navigation	790
Table of content navigation	790
Keyboard navigation with Tab and Shift+Tab keys	793
See also	794
Magnification in Angular PDF Viewer component.....	794
See also	796
Text Search in Angular PDF Viewer component	796
See also	798
Annotation	798
Text Markup Annotation in the Angular PDF Viewer component.....	798
Shape Annotation in Angular PDF Viewer component	823
Stamp Annotation in Angular PDF Viewer component	836
Sticky Notes Annotation in the Angular PDF Viewer component.....	848
Measurement Annotation in Angular PDF Viewer component.....	857
Free text annotation	873
Ink Annotation in Angular PDF Viewer component.....	885
Import and export annotation	902
Comments.....	918
Handwritten Signature.....	923
Adding a handwritten signature to the PDF document.....	923
Editing the properties of handwritten signature	924
Interaction Mode in Angular PDF Viewer component.....	925
Selection mode	925
Panning Mode.....	926
See also	927

Form Designer	927
Create form fields programmatically	927
Create form fields with UI interaction	962
Print in Angular PDF Viewer component	967
Print the PDF document in the new window	969
Limiting the Dialog Opening for Printing	970
See also	971
Download in Angular PDF Viewer component	972
See also	973
Localization in Angular PDF Viewer component	973
Accessibility in Syncfusion Angular PDF Viewer components	979
WAI-ARIA attributes	980
Keyboard interaction	980
Ensuring accessibility	984
See also	985
How To	985
Customize the toolbar	985
magnificationToolbar {	987
magnificationToolbar.e-toolbar .e-toolbar-items {	987
magnificationToolbar.e-toolbar .e-tbar-btn {	987
topToolbar {	987
popup .e-dlg-content {	988
Unload the PDF document programmatically	997
Load PDF documents dynamically	997
Include the Authorization token	998
Get the Base 64 string of the loaded PDF document	999
Customize the selection border	1000
Extract Text	1001
Import and Export annotation as object	1002
Delete a specific annotation using deleteAnnotationById	1002
Open Thumbnail pane programmatically	1003
How to enable and disable the delete button based on annotation selection and unselection events	1004
Add the custom stamp based on the free text bounds	1005
Install packages required for versions below 12	1005

Load Microsoft Office files	1006
How to Change the Font Family in the Type Signature	1009
Resolve "Unable to find an entry point named FPDFText_GetCharAngle" error	1010
How to clear the "Web-service is not listening" to error	1011
Load N number of pages on initial loading	1013
Retry Timeout	1014
Configure Redis Cache	1015
Customize toolbar in PDF Viewer component.....	1017
Know the supported conformance PDF documents in Anglar PDF Viewer component	1021
Organize Pages Feature	1022
Customize context menu	1023
PageRenderInitiate and PageRenderComplete event	1034
Open and Close Bookmark pane programmatically	1035
Locking Form Fields in a PDF document	1036
Troubleshooting.....	1041
Experience the Standalone PDF Viewer Component by copying assets from node_modules into my app	1041
Troubleshoot error 'cp' is not recognized as a command	1041
Document Loading Issues in Version 23.1 or Newer	1042
Uncaught DOMException: Failed to execute 'importScripts' on 'WorkerGlobalScope'	1043

Kanban

Getting started with Angular Kanban component

This section explains the steps required to create a simple Angular Kanban component and configure its available functionalities.

Setup Angular Environment

Use [Angular CLI](#) to setup the Angular applications. To install Angular CLI, use the following command.

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

Create an Angular Application

Start a new Angular application using the following Angular CLI command.

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

Installing Syncfusion Kanban package

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

Currently, Syncfusion provides two types of package structures for Angular components. They are:

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

Ivy library distribution package

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

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

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

Angular compatibility compiled package(ngcc)

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

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

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

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

```
`bash
@syncfusion/ej2-angular-kanban:"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 Kanban Module

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

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

Adding CSS reference

Add Kanban component's styles as given in the following `styles.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/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-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-angular-kanban/styles/material.css';
,
```

Add Kanban component

Modify the template in the [src/app/app.component.ts] file to render the Kanban component. Add the Angular Kanban by using the `<ejs-kanban>` selector in the `template` section of the app.component.ts file.

src/app/app.component.ts

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the Kanban component
  template: `<ejs-kanban>
<e-columns>
<e-column headerText='To do' keyField='Open'></e-column>
<e-column headerText='In Progress' keyField='InProgress'></e-column>
<e-column headerText='Testing' keyField='Testing'></e-column>
<e-column headerText='Done' keyField='Close'></e-column>
</e-columns>
</ejs-kanban>`
})
export class AppComponent { }
```

Run the application

Use the following command to run the application in the browser.

```
`bash
ng serve --open
,
```

The output will appear as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
@Component({
```



```

imports: [
    KanbanModule
],
standalone: true,
selector: 'app-root',
template: `<ejs-kanban>
    <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
</ejs-kanban>`
})
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));

```

Populating cards

To populate the empty Kanban with cards, define the local JSON data or remote data using the **dataSource** property. To define **dataSource**, the mandatory fields in JSON object should be relevant to **keyField**. In the following example, you can see the cards defined with default fields such as ID, Summary, and Status.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
imports: [

    KanbanModule
],
standalone: true,
selector: 'app-root',
template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
        <e-column headerText='Testing' keyField='Testing'></e-
column>

```

```

        <e-column headerText='Done' keyField='Close'></e-column>
      </e-columns>
    </ejs-kanban>`
  })
  export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
      contentField: 'Summary',
      headerField: 'Id'
    };
  }
}

```

MAIN.TS

```

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

```

Enable swimlane

Swimlane can be enabled by mapping the fields `swimlaneSettings.keyField` to appropriate column name in `dataSource`. This enables the grouping of the cards based on the mapped column values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
      <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
      </e-columns>
    </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',

```

```

        headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = { keyField: 'Assignee'
};
}

```

MAIN.TS

```

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

```

Columns in Angular Kanban component

The **Kanban** columns represent the each stage of the process. The column definitions are used as the **dataSource** schema in the Kanban. The Kanban operations such as drag-and-drop, swimlane, and toggle columns are performed based on column definitions.

Single-key mapping

Kanban columns are categorized by mapping the **key** from the datasource using the **keyField** property. The corresponding **value** in the datasource is mapped inside the columns **keyField**. Based on this categorization, Kanban columns are split on this board.

The **keyField** property is mandatory to render the columns in the Kanban board.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {

```

```

        contentField: 'Summary',
        headerField: 'Id'
    };
}

```

MAIN.TS

```

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

```

Multi-key mapping

Kanban board allows to render a single column by mapping multiple keys using `keyField` property. In below sample, specified the multiple keys(Open, Validate) to a single column.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open,
Validate'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}

```

MAIN.TS

```

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

```

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

Header text

You can provide the column header text of Kanban columns using the `headerText` property. If you have not specified any header text, it will render the header without any text.

Header template

You can customize the column header with any HTML or CSS element using the `template` property as shown in the following code.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, ColumnsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' cssClass='kanban-header-template'
[dataSource]='data' [cardSettings]='cardSettings'>
    <e-columns>
      <e-column *ngFor="let column of columns;"
headerText={{column.headerText}} keyField='{{column.keyField}}'>
        <ng-template #template let-data>
          <div class="header-template-wrap">
            <div class="header-icon e-icons"
{{data.keyField}}"></div>
            <div class="header-
text">{{data.headerText}}</div>
          </div>
        </ng-template>
      </e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    headerField: 'Id',
    contentField: 'Summary'
  };
  public columns: ColumnsModel[] = [
    { headerText: 'To Do', keyField: 'Open' },
    { headerText: 'In Progress', keyField: 'InProgress' },
    { headerText: 'In Review', keyField: 'Review' },
```

```

    { headerText: 'Done', keyField: 'Close' }
  ];
}

```

MAIN.TS

```

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

```

Toggle columns

Kanban allows to expand or collapse its columns using `allowToggle` inside the `columns` property. When enable the property, it will render the expand or collapse icon to the column header.

By default, collapsed column width is set to `50px`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'
allowToggle='true'></e-column>
      <e-column headerText='In Progress' keyField='InProgress'
allowToggle='true'></e-column>
      <e-column headerText='Testing' keyField='Testing'
allowToggle='true'></e-column>
      <e-column headerText='Done' keyField='Close'
allowToggle='true'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}

```

MAIN.TS

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

Initially collapsed column

By default, all columns are on expanded state when loading the Kanban board initially. But, you can render the columns with collapsed state using the `isExpanded` property.

The `isExpanded` property only works when enabling the `allowToggle` property on particular column.

In the following example, the backlog column is collapsed on initialization of Kanban board.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'
allowToggle='true' [isExpanded]='isExpanded'></e-column>
      <e-column headerText='In Progress' keyField='InProgress'
allowToggle='true'></e-column>
      <e-column headerText='Testing' keyField='Testing'
allowToggle='true' [isExpanded]='isExpanded'></e-column>
      <e-column headerText='Done' keyField='Close'
allowToggle='true'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public isExpanded: Boolean = false;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}
```

MAIN.TS

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

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

Stacked headers

Stacked headers are the additional headers to column header that will group the similar columns.

Define the grouping of columns **key** value to the **keyFields** property and provide the custom header text name to grouped columns using the **text** property, which is placed inside the **stackedHeaders** property.

In the following code, the kanban columns 'InProgress, Review' are grouped under 'Development Phase' category.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='Open' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Review' keyField='Review'></e-
column>
      <e-column headerText='Completed' keyField='Close'></e-
column>
    </e-columns>
    <e-stackedHeaders>
      <e-stackedHeader text='To Do' keyFields='Open'></e-
stackedHeader>
      <e-stackedHeader text='Development Phase'
keyFields='InProgress, Review'></e-stackedHeader>
      <e-stackedHeader text='Done' keyFields='Close'></e-
stackedHeader>
    </e-stackedHeaders>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
};
```



```
}

```

MAIN.TS

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

Cards in Angular Kanban component

The cards are main elements in Kanban board, which represent the task information with header and content. The header and content of a card is fetched from the corresponding mapping fields. The card layout can be customized with template also.

Drag-and-drop

Transit or change the card position using the drag-and-drop functionality. By default, the `allowDragAndDrop` property is enabled on the Kanban board, which is used to change the card position by column-to-column or within the column.

Added dotted border on Kanban cells except the dragged clone cells when dragging, which indicates the possible ways for dropping the cards into the cells.

Header

The card header is achieved by mapping the `headerField` property, which is placed inside the `cardSettings` property. By default, the `showHeader` property enabled by Kanban board that is used to show the header at the top of the card.

The `headerField` property of `cardSettings` is mandatory to render the cards in the Kanban board. It acts as a unique field that is used to avoid the duplication of card data. You can not change the `headerField` of mapped data value using the `updateCard` public method or server-side update of data.

In the following demo, the `showHeader` property is disabled on Kanban board.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
        keyField='InProgress'></e-column>
```

```

        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
</ejs-kanban>`
}))
export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id',
        showHeader: false
    };
}

```

MAIN.TS

```

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

```

Content

The card's content is fetched from data source using the `contentField` property, which is placed inside the `cardSettings` property. If the `contentField` property is not used, card is rendered with empty content.

Template

You can customize the default card layout using template as per your application needs. This can be achieved by template of the `cardSettings` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
        <ng-template #cardSettingsTemplate let-data>

```

```

        <div class='e-card-content'>
            <table class="card-template-wrap">
                <tbody>
                    <tr>
                        <td class="CardHeader">Id:</td>
                        <td>{{data.Id}}</td>
                    </tr>
                    <tr>
                        <td class="CardHeader">Type:</td>
                        <td>{{data.Type}}</td>
                    </tr>
                    <tr>
                        <td class="CardHeader">Priority:</td>
                        <td>{{data.Priority}}</td>
                    </tr>
                    <tr>
                        <td class="CardHeader">Summary:</td>
                        <td>{{data.Summary}}</td>
                    </tr>
                </tbody>
            </table>
        </div>
    </ng-template>
</ejs-kanban>`
    ))
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            headerField: 'Id'
        };
    }
}

```

MAIN.TS

```

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

```

Selection

Kanban board allows to select single and multiple selection of cards when mouse or keyboard interactions using **selectionType** property. The property contains following types.

- **None:** No cards are allowed to select from Kanban board.
- **Single:** Only one card allowed to select at a time in the Kanban board.
- **Multiple:** Multiple cards are allowed to select in a board.

Multiple Selection

Select the multiple cards randomly using Ctrl + mouse click and select the multiple cards continuously using Shift + mouse click action on Kanban board. Set **Multiple** in **selectionType** to enable the multiple selection in a board.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='Backlog' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    headerField: 'Id',
    contentField: 'Summary',
    selectionType: 'Multiple'
  };
}

```

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 Kanban component

The Kanban uses **DataManager**, which supports both RESTful data service binding and JavaScript object array binding. The [dataSource](#) property of Kanban can be assigned either with the instance of **DataManager** or JavaScript object array collection, as it supports the following two data binding methods:

- Local data
- Remote data

Local data

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

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}
```

MAIN.TS

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

By default, [DataManager](#) uses [JsonAdaptor](#) for binding local data.

Remote data

To bind remote data to kanban component, assign service data as an instance of [DataManager](#) to the [dataSource](#) property. To interact with remote data source, provide the endpoint [url](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='dataManager'
[cardSettings]='cardSettings' [allowDragAndDrop]='false'
(dialogOpen)="dialogOpen($event)">
      <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
      </e-columns>
    </ejs-kanban>`
})
export class AppComponent {
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public dataManager: DataManager = new DataManager({
    url: 'https://ej2services.syncfusion.com/production/web-
services/api/Kanban',
    adaptor: new ODataAdaptor,
    crossDomain: true
  });
  public dialogOpen(args: DialogEventArgs): void {
    args.cancel = true;
  }
}

```

MAIN.TS

```

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

```

By default, [DataManager](#) uses **ODataAdaptor** for remote data-binding.

OData services

[OData](#) is a standardized protocol for creating and consuming data. You can retrieve data from OData service using the DataManager. Refer to the following code example for remote Data binding using OData service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    KanbanModule
  ],
  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));
```

OData v4 services

The ODataV4 is an improved version of OData protocols, and the [DataManager](#) can also retrieve and consume OData v4 services. For more details on OData v4 services, refer to the [odatadocumentation](#). To bind OData v4 service, use the **ODataV4Adaptor**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='ContactTitle' [dataSource]='dataManager'
[cardSettings]='cardSettings' [allowDragAndDrop]='false'
(dialogOpen)="dialogOpen($event)">
      <e-columns>
        <e-column headerText='Order Administrator' keyField='Order
Administrator'></e-column>
        <e-column headerText='Sales Representative'
keyField='Sales Representative'></e-column>
        <e-column headerText='Export Administrator'
keyField='Export Administrator'></e-column>
      </e-columns>
    </ejs-kanban>`
})
export class AppComponent {
  public cardSettings: CardSettingsModel = {
    contentField: 'ContactName',
    headerField: 'SupplierID'
  };
  public dataManager: DataManager = new DataManager({
    url:
'https://services.odata.org/v4/northwind/northwind.svc/Suppliers',
    adaptor: new ODataV4Adaptor,
  });
  public dialogOpen(args: DialogEventArgs): void {
    args.cancel = true;
  }
}
```

MAIN.TS

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



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

Web API

You can use **WebApiAdaptor** to bind kanban with Web API created using OData endpoint.

```
`typescript
```

```
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='dataManager' [cardSettings]='cardSettings'
[allowDragAndDrop]='false' (dialogOpen)="dialogOpen($event)">
<e-columns>
<e-column headerText='To do' keyField='Open'></e-column>
<e-column headerText='In Progress' keyField='InProgress'></e-column>
<e-column headerText='Testing' keyField='Testing'></e-column>
<e-column headerText='Done' keyField='Close'></e-column>
</e-columns>
</ejs-kanban>`
})
export class AppComponent {
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  private dataManager: DataManager = new DataManager({
    url: '/api/Tasks',
    adaptor: new WebApiAdaptor
  });
  public dialogOpen(args: DialogEventArgs): void {
    args.cancel = true;
  }
}
```

Below server-side controller code to get the Kanban data.

```
`typescript
[HttpGet]
public object Get()
{
    var data = OrdersDetails.GetAllRecords().ToList();
    return data;
}
```

URL adaptor

The CRUD (Create, Read, Update and Delete) actions can be performed easily on Kanban cards using the various adaptors available within the **DataManager**. Most preferably, we will be using **UrlAdaptor** for performing CRUD actions on Kanban.

The CRUD operation in Kanban can be mapped to server-side controller actions using the properties **insertUrl**, **removeUrl**, **updateUrl**, and **crudUrl**.

- **insertUrl** – You can perform a single insertion operation on the server-side.
- **updateUrl** – You can update single data on the server-side.
- **removeUrl** – You can remove single data on the server-side.
- **crudUrl** – You can perform bulk data operation on the server-side.

```
`typescript
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
@Component({
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='dataManager' [cardSettings]='cardSettings'>
    <e-columns>
    <e-column headerText='To do' keyField='Open'></e-column>
    <e-column headerText='In Progress' keyField='InProgress'></e-column>
    <e-column headerText='Testing' keyField='Testing'></e-column>
    <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
    </ejs-kanban>`
})
```

```
export class AppComponent {  
  public cardSettings: CardSettingsModel = {  
    contentField: 'Summary',  
    headerField: 'Id'  
  };  
  private dataManager: DataManager = new DataManager({  
    url: 'Home/DataSource',  
    updateUrl: 'Home/Update',  
    insertUrl: 'Home/Insert',  
    removeUrl: 'Home/Delete',  
    adaptor: new UrlAdaptor(),  
    crossDomain: true  
  });  
}
```

The server-side controller code to handle the CRUD operations are as follows.

```
`typescript  
private NORTHWNDEntities db = new NORTHWNDEntities();  
public ActionResult DataSource() {  
  var DataSource = db.Tasks.ToList();  
  return Json(DataSource, JsonRequestBehavior.AllowGet);  
}  
public ActionResult Insert(Params value) {  
  //Insert card data into the database  
  return Json(value, JsonRequestBehavior.AllowGet);  
}  
public ActionResult Update(Params value) {  
  //Update card data into the database  
  return Json(value, JsonRequestBehavior.AllowGet);  
}  
public void Delete(Params value) {  
  //Delete card data from the database  
}
```

```

public class Params {
public int Id { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}

```

Custom adaptor

It is possible to create your own custom adaptor by extending the built-in available adaptors. The following example demonstrates the custom adaptor usage and how to add a custom field **TaskId** for the cards by overriding the built-in response processing using the **processResponse** method of the **ODataAdaptor**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, ODataAdaptor } from '@syncfusion/ej2-data';
class TaskIdAdaptor extends ODataAdaptor {
  override processResponse(): Object {
    let i = 0;
    // calling base class processResponse function
    let original: any = super.processResponse.apply(this, arguments as any);
    // adding Task Id
    original.forEach((item: any) => item['Id'] = 'Task - ' + ++i);
    return original;
  }
}
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='dataManager'
[cardSettings]='cardSettings' [allowDragAndDrop]='false'
(dialogOpen)="dialogOpen($event)">
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`

```

```

    })
    export class AppComponent {
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public dataManager: DataManager = new DataManager({
            url: 'https://ej2services.syncfusion.com/production/web-
            services/api/Kanban',
            adaptor: new TaskIdAdaptor
        });
        public dialogOpen(args: DialogEventArgs): void {
            args.cancel = true;
        }
    }
}

```

MAIN.TS

```

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

```

Sending additional parameters to the server

To add a custom parameter to the data request, use the **addParams** method of **Query** class. Assign the **Query** object with additional parameters to the kanban [query](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { DataManager, ODataAdaptor, Query } from '@syncfusion/ej2-data';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='dataManager'
    [cardSettings]='cardSettings' [allowDragAndDrop]='false'
    (dialogOpen)="dialogOpen($event)" [query]='query'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
            keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
            column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`

```

```

    })
    export class AppComponent {
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public dataManager: DataManager = new DataManager({
            url: 'https://ej2services.syncfusion.com/production/web-
services/api/Kanban',
            adaptor: new ODataAdaptor
        });
        public query: Query = new Query().addParams('ej2kanban', 'true');
        public dialogOpen(args: DialogEventArgs): void {
            args.cancel = true;
        }
    }
}

```

MAIN.TS

```

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

```

The parameters added using the [query](#) property will be sent along with the data request for every kanban action.

Handling HTTP error

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

The argument passed to the [actionFailure](#) event contains the error details returned from the server.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component, ViewChild } from '@angular/core';
import { CardSettingsModel, KanbanComponent } from '@syncfusion/ej2-angular-kanban';
import { DataManager } from '@syncfusion/ej2-data';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban #Kanban keyField='Status'
[dataSource]='dataManager' [cardSettings]='cardSettings'
(actionFailure)="onActionFailure($event)">
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>

```

```

        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
</ejs-kanban>`
    })
    export class AppComponent {
        @ViewChild('Kanban') public kanban?: KanbanComponent;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public dataManager: DataManager = new DataManager({
            url: 'http://some.com/invalidUrl'
        });
        onActionFailure(e: Error): void {
            let span: HTMLElement = document.createElement('span');
            ((this.kanban as KanbanComponent).element.parentNode as
            ParentNode).insertBefore(span, (this.kanban as KanbanComponent).element);
            span.style.color = '#FF0000'
            span.innerHTML = 'Server exception: 404 Not found';
        }
    }
}

```

MAIN.TS

```

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

```

The [actionFailure](#) event will be triggered not only for the server errors, but also when there is an exception while processing the kanban actions.

Loading data via ajax

You can use Kanban [dataSource](#) property to bind the datasource to Kanban from external ajax request. In the following code, we have fetched the datasource from the server using ajax request and provided that to the [dataSource](#) property by using the **onSuccess** event of ajax.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component, ViewChild } from '@angular/core';
import { KanbanComponent, CardSettingsModel, DialogEventArgs } from
'@syncfusion/ej2-angular-kanban';
import { Ajax } from '@syncfusion/ej2-base';
import { DataManager } from '@syncfusion/ej2-data';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template: `<input type="button" id="btn" (click)="click()" value="Click"/>
    <ejs-kanban #Kanban keyField='ShipCountry' [dataSource]='dataManager'
    [cardSettings]='cardSettings' (actionFailure)="onActionFailure($event)">
        <e-columns>
            <e-column headerText='Denmark' keyField='Denmark'></e-
column>
            <e-column headerText='Brazil' keyField='Brazil'></e-
column>
            <e-column headerText='Switzerland'
keyField='Switzerland'></e-column>
            <e-column headerText='Germany' keyField='Germany'></e-
column>
        </e-columns>
    </ejs-kanban>`
  })
  export class AppComponent {
    onActionFailure($event: any) {
      throw new Error('Method not implemented.');
```

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

Observables in Angular Kanban 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

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

```
`ts
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { TaskService } from './task.service';
import { DataStateChangeEventArgs, DataSourceChangedEventArgs } from '@syncfusion/ej2-kanban';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';

@Component({
  selector: 'app-root',
  template: `<ejs-kanban #kanbanObj keyField='Status' [dataSource]='data | async'
    [cardSettings]='cardSettings' (dataStateChange)= 'dataStateChange($event)'
    (dataSourceChanged)='dataSourceChanged($event)'>
    <e-columns>
    <e-column headerText='To Do' keyField='Open'></e-column>
    <e-column headerText='In Progress' keyField='InProgress'></e-column>
    <e-column headerText='Testing' keyField='Testing'></e-column>
    <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`,
  providers: [TaskService]
})
export class AppComponent implements OnInit {
  public data: Observable<DataStateChangeEventArgs>;
  public state: DataStateChangeEventArgs;
  public cardSettings: CardSettingsModel;
  constructor(private service: TaskService) {
    this.data = service;
  }
  public dataSourceChanged(state: DataSourceChangedEventArgs): void {
    if (state.requestType === 'cardCreated') { state.endEdit() }
  }
}
```

```

else if (state.requestType === 'cardChanged') { state.endEdit() }
else if (state.requestType === 'cardRemoved') { state.endEdit() }
}
public dataStateChange(state: DataStateChangeEventArgs): void {
this.service.execute(state);
}
public ngOnInit(): void {
let state = { skip: 0, take: 10 };
this.service.execute(state);
this.cardSettings = {
contentField: 'Summary',
headerField: 'Id'
};
}
}
`ts
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { DataStateChangeEventArgs } from '@syncfusion/ej2-angular-kanban'
import { Observable } from 'rxjs';
import { Subject } from 'rxjs';
import { map } from 'rxjs/operators';
@Injectable()
export class TasksService extends Subject<DataStateChangeEventArgs> {
private BASE_URL = 'https://ej2services.syncfusion.com/production/web-services/api/Kanban';
constructor(private http: Http) {
super();
}
public execute(state: any): void {
this.getData(state).subscribe(x => super.next(x));
}
protected getData(state: DataStateChangeEventArgs): Observable<DataStateChangeEventArgs> {

```

```

return this.http
.get(`${this.BASE_URL}`)
.pipe(map((response: any) => response.json()))
.pipe(map((response: any) => (<any>{
result: response
})))
.pipe((data: any) => data);
}
}
`

```

You should maintain the same [Observable](#) instance for every kanban action.

When initial rendering, the `dataStateChange` event will not be triggered. You can perform the operation in the `ngOnInit` if you want the kanban to show the cards.

Perform CRUD operations

The `dataSourceChanged` event is triggered to update the kanban data. You can perform the save operation based on the event arguments and you need to call the `endEdit` method to indicate the completion of the save operation.

```

`ts
import { Component, OnInit } from '@angular/core';
import { Observable } from 'rxjs';
import { TaskService } from './task.service';
import { DataStateChangeEventArgs, DataSourceChangedEventArgs } from '@syncfusion/ej2-kanban';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
@Component({
selector: 'app-root',
template: `
<ejs-kanban #kanbanObj keyField='Status' [dataSource]='data | async' [cardSettings]='cardSettings'
(dataStateChange)= 'dataStateChange($event)' (dataSourceChanged)= 'dataSourceChanged($event)'>
<e-columns>
<e-column headerText='To Do' keyField='Open'></e-column>
<e-column headerText='In Progress' keyField='InProgress'></e-column>
<e-column headerText='Testing' keyField='Testing'></e-column>
<e-column headerText='Done' keyField='Close'></e-column>
</e-columns>

```

```
</ejs-kanban>`,
})
export class AppComponent implements OnInit {
  public data: Observable<DataStateChangeEventArgs>;
  public state: DataStateChangeEventArgs;
  public cardSettings: CardSettingsModel;
  constructor(private service: TasksService) {
    this.data = service;
  }
  public dataSourceChanged(state: DataSourceChangedEventArgs): void {
    if (state.requestType === 'cardCreated') {
      this.crudService.addCard(state).subscribe(() => {
        state.endEdit();
      });
    } else if (state.requestType === 'cardChanged') {
      this.crudService.updateCard(state).subscribe(() => {
        state.endEdit();
      });
    } else if (state.requestType === 'cardRemoved') {
      this.crudService.deleteCard(state).subscribe(() => {
        state.endEdit();
      });
    }
  }
  public dataStateChange(state: DataStateChangeEventArgs): void {
    this.service.execute(state);
  }
  public ngOnInit(): void {
    let state = { skip: 0, take: 10 };
    this.service.execute(state);
    this.cardSettings = {
      contentField: 'Summary',
      headerField: 'Id'
    }
  }
}
```

```

};
}
}
`ts
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';
import { DataStateChangeEventArgs } from '@syncfusion/ej2-angular-kanban'
import { Observable } from 'rxjs';
import { Subject } from 'rxjs';
import { map } from 'rxjs/operators';
@Injectable()
export class TasksService extends Subject<DataStateChangeEventArgs> {
  private BASE_URL = 'https://ej2services.syncfusion.com/production/web-services/api/Kanban';
  constructor(private http: Http) {
    super();
  }
  public execute(state: any): void {
    this.getData(state).subscribe(x => super.next(x));
  }
  protected getData(state: DataStateChangeEventArgs): Observable<DataStateChangeEventArgs> {
    return this.http
      .get(`${this.BASE_URL}`)
      .pipe(map((response: any) => response.json()))
      .pipe(map((response: any) => (<any>{
        result: response
      })))
      .pipe((data: any) => data);
  }
  / POST: add a new record to the server */
  addCard(state: DataSourceChangedEventArgs): Observable<Customer> {
    return this.http.post<Customer>(this.customersUrl, state.addedRecords[0], httpOptions);
  }
}

```

```

/ DELETE: delete the record from the server */
deleteCard(state: any): Observable<Customer> {
  const id = state.deletedRecords[0].taskId;
  const url = `${this.customersUrl}/${id}`;
  return this.http.delete<Customer>(url, httpOptions);
}

/ PUT: update the record on the server */
updateCard(state: DataSourceChangedEventArgs): Observable<any> {
  return this.http.put(this.customersUrl, state.changedRecords[0], httpOptions);
}
}
,

```

Swimlane in Angular Kanban component

Swimlanes are horizontal categorizations of cards on the Kanban board. It is used for grouping of cards, which brings transparency to the workflow process.

Render swimlane row

Cards can be grouped based on `keyField` and displayed in rows, which are separated by columns. It is mandatory to define the `keyField` that is mapped from the datasource for rendering swimlane rows in the Kanban board.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>

```

```

        </ejs-kanban>`
    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public swimlaneSettings: SwimlaneSettingsModel = { keyField: 'Assignee'
    };
    }
}

```

MAIN.TS

```

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

```

Custom row text

Customize the swimlane row header text by using the `textField` property mapped from datasource.

It is not mandatory to define the `textField` to `swimlaneSettings`. It will automatically consider the `keyField` to swimlane row header text.

If the mapping `textField` key is not present in the datasource, it will consider the swimlane `keyField` as swimlane row header text.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
})

```

```

    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public swimlaneSettings: SwimlaneSettingsModel = {
            keyField: 'Assignee',
            textField: 'AssigneeName'
        };
    }
}

```

MAIN.TS

```

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

```

Template

You can customize the Kanban swimlane row by using the `template` property, which is specified within the `swimlaneSettings` property. In this demo, the swimlane header is customized with HTML element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
            keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
            column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
        <ng-template #swimlaneSettingsTemplate let-data>
            <div id="swimlaneTemplate">
                <div class="swimlane-template e-swimlane-template-
                table">
                    <div class="e-swimlane-row-text">

```



```

                <span>{{data.textField}}</span>
            </div>
        </div>
    </div>
</ng-template>
</ejs-kanban>`
))
export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = {
        keyField: 'Assignee',
        textField: 'AssigneeName'
    };
}

```

MAIN.TS

```

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

```

Sorting

Swimlane rows are rendered on descending order when using the `sortBy` property set to `Descending` order. By default, swimlane rows are rendered by `Ascending` order.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
    <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
    </e-columns>
</template>

```

```

        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
</ejs-kanban>`
}))
export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = {
        keyField: 'Assignee',
        sortBy: 'Descending'
    } as SwimlaneSettingsModel;
}

```

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

By default, The Kanban does not allow dragging the cards across the swimlane rows. Enabling the `dragAndDrop` property allows you to drag the cards across the swimlane rows, which is specified inside `swimlaneSettings` property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>

```

```

        </e-columns>
    </ejs-kanban>`
    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        public swimlaneSettings: SwimlaneSettingsModel = {
            keyField: 'Assignee',
            allowDragAndDrop: true
        };
    }
}

```

MAIN.TS

```

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

```

Create empty row

You can render the empty swimlane row by enabling the `showEmptyRow` property. If mapping `keyField` does not have cards, empty swimlane row will be rendered.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
            keyField='InProgress'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {

```

```

        contentField: 'Summary',
        headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = {
        keyField: 'Assignee',
        showEmptyRow: true
    };
}

```

MAIN.TS

```

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

```

Calculate cards count

Users can show or hide the cards count by swimlane row in header when enabling the `showItemCount` property, which is enabled by default on the Kanban board.

Provided localization support for `items` text.

In below demo, disabled on `showItemCount` property on rendering swimlane row without total count.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
            keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
            column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
})
export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',

```

```

        headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = {
        keyField: 'Assignee',
        showItemCount: false
    };
}

```

MAIN.TS

```

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

```

Enable frozen rows

Frozen rows provide an option to make the current swimlane row header text always visible on top of the content while scrolling the Kanban content. The swimlane header text will be changed dynamically, when you scroll to another swimlane row.

By default, the `enableFrozenRows` property is set as `false`. If you wish to show the swimlane frozen rows, you can enable the `enableFrozenRows` property.

This feature support only when using Kanban content scrolling.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'
    height='500px'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
})
export class AppComponent {

```

```
public data: Object[] = kanbanData;
public cardSettings: CardSettingsModel = {
  contentField: 'Summary',
  headerField: 'Id'
};
public swimlaneSettings: SwimlaneSettingsModel = {
  keyField: 'Assignee',
  enableFrozenRows: true
};
}
```

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 in Angular Kanban component

All cards can be dragged and dropped across the columns or within the columns or swimlane row or kanban to an external source and vice versa.

The following drag and drop types are available in the Kanban board.

- Internal drag and drop
- Column drag and drop
- Swimlane drag and drop
- External drag and drop
- Kanban to Kanban
- Kanban to External source and vice versa.

Dropped card position varies based on the `sortSettings` property.

Internal drag and drop

Allows the user to drag and drop the cards within the kanban board. Based on this, we can categorize into two ways.

- Column drag and drop
- Swimlane drag and drop

Column drag and drop

By default, all cards can be dragged and dropped across the columns and within the columns. You cannot drag and drop the cards when disabling the `allowDragAndDrop` property.

You can prevent the drag or drop behavior of the particular column by disabling the `allowDrag` or `allowDrop` property.

You can also control the flow of transition cards between the columns by using the `transitionColumns` property.

In the following example, disable the drag and drop behavior on the Kanban board.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [allowDragAndDrop]='allowDragAndDrop'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public allowDragAndDrop: Boolean = false;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
};
}
```

MAIN.TS

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

Swimlane drag and drop

By default, Swimlane allows drag and drop across the columns within the swimlane row. Kanban does not allow dragging the cards across the swimlane rows.

Enabling the `dragAndDrop` property allows you to drag the cards across the swimlane rows, which is specified inside the `swimlaneSettings` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
```

```
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public swimlaneSettings: SwimlaneSettingsModel = { keyField: 'Assignee',
allowDragAndDrop: true };
}
```

MAIN.TS

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

External drag and drop

Allows the user to drag and drop the cards from one kanban to another kanban or Kanban to an external source and vice versa.

Kanban to kanban

Drag and drop the card from one kanban to another kanban and vice versa. This can be achieved by specifying the `externalDropId` property which is used to specify the id of the dropped kanban element and the `dragStop` event which is used to delete the card on dragged Kanban and add the card on dropped Kanban using the `deleteCard` and `addCard` public methods.

Before adding a card to dropped kanban, you can manually change the card data `headerField` when the same card data `headerField` is dropped to another Kanban.

In the following example, Drag the card from one Kanban and drop it into another kanban using the `dragStop` event. In this event, remove the card from the dragged Kanban by using the `deleteCard` public method and add the card to the dropped Kanban by using the `addCard` public method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component, ViewChild } from '@angular/core';
import { closest } from '@syncfusion/ej2-base';
import { CardSettingsModel, DragEventArgs, KanbanComponent } from
'@syncfusion/ej2-angular-kanban';
import { kanbanAData, kanbanBData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="container-fluid">
    <div class="row">
      <div class="col-sm-6">
        <h4>Kanban A</h4>
        <ejs-kanban id='KanbanA' #KanbanA keyField='Status'
[dataSource]='dataA' [externalDropId]='externalKanbanADropId'
[cardSettings]='cardSettings'
(dragStop)='onKanbanADragStop($event)'>
          <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
          </e-columns>
        </ejs-kanban>
      </div>
      <div class="col-sm-6">
        <h4>Kanban B</h4>
        <ejs-kanban id='KanbanB' #KanbanB keyField='Status'
[dataSource]='dataB' [externalDropId]='externalKanbanBDropId'
[cardSettings]='cardSettings'
(dragStop)='onKanbanBDragStop($event)'>
          <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
          </e-columns>
        </ejs-kanban>
      </div>
    </div>`
  })
export class AppComponent {
  @ViewChild('KanbanA')
  public kanbanObjA?: KanbanComponent;
  @ViewChild('KanbanB')
  public kanbanObjB?: KanbanComponent;
  public dataA: Object[] = kanbanAData;
  public dataB: Object[] = kanbanBData;
```

```

    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id'
    };
    public externalKanbanADropId: string[] = ["#KanbanB"];
    public externalKanbanBDropId: string[] = ["#KanbanA"];
    onKanbanADragStop(args: DragEventArgs) {
        let kanbanBElement: Element = <Element>closest(args.event.target as
Element, '#KanbanB');
        if (kanbanBElement) {
            (this.kanbanObjA as KanbanComponent).deleteCard(args.data);
            (this.kanbanObjB as KanbanComponent).addCard(args.data,
args.dropIndex);
            args.cancel = true;
        }
    };
    onKanbanBDragStop(args: DragEventArgs) {
        let kanbanAElement: Element = <Element>closest(args.event.target as
Element, '#KanbanA');
        if (kanbanAElement) {
            (this.kanbanObjB as KanbanComponent).deleteCard(args.data);
            (this.kanbanObjA as KanbanComponent).addCard(args.data,
args.dropIndex);
            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));

```

Treeview to Kanban

Drag the card from the Kanban board and drop it to the Treeview component and vice versa.

In the following sample, remove the data from the Kanban board using the `deleteCard` public method and add to the Treeview component using the `addNodes` public method at Kanban `dragStop` event when dragging the card and dropping it to the Treeview component. Remove the data from Treeview using the `removeNodes` public method and add to Kanban board using the `openDialog` public method when dragging the list from the Treeview component and dropping it to the kanban board.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { TreeViewModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { closest } from '@syncfusion/ej2-base';
import { CardSettingsModel, DragEventArgs, KanbanComponent } from
'@syncfusion/ej2-angular-kanban';
import { kanbanData, treeViewData } from './datasource';

```

```

import { TreeViewComponent } from '@syncfusion/ej2-angular-navigations';
import { DragAndDropEventArgs } from '@syncfusion/ej2-navigations';
@Component({
  imports: [

    KanbanModule,
    TreeViewModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="container-fluid">
    <div class="row">
      <div class="col-md-6">
        <h4>Kanban</h4>
        <ejs-kanban id='Kanban' #Kanban keyField='Status'
[dataSource]='data' [externalDropId]='externalKanbanDropId'
[cardSettings]='cardSettings'
(dragStop)='onKanbanDragStop($event)'>
          <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
          </e-columns>
        </ejs-kanban>
      </div>
      <div class="col-md-6">
        <h4>TreeView</h4>
        <ejs-treeview id='TreeView' #TreeView [fields]='field'
[allowDragAndDrop]='allowDragAndDrop'
(nodeDragStop)="onTreeDragStop($event)">
          <ng-template #nodeTemplate="" let-data="">
            <div id="treelist">
              <div id="treeviewlist">{{data.Id}} - {{data.Status}}</div>
            </div>
          </ng-template>
        </ejs-treeview>
      </div>
    </div>`
})
export class AppComponent {
  @ViewChild('Kanban')
  public kanbanObj?: KanbanComponent;
  @ViewChild('TreeView')
  public treeObj?: TreeViewComponent;
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public externalKanbanDropId: string[] = ['#TreeView'];
  public field: Object = { dataSource: treeViewData, id: 'Id', text:
'Status' };
  public allowDragAndDrop: boolean = true;
  onKanbanDragStop(args: DragEventArgs) {
    let treeViewElement: Element = <Element>closest(args.event.target as
Element, '#TreeView');
    if (treeViewElement) {

```

```

        (this.kanbanObj as KanbanComponent).deleteCard(args.data);
        (this.treeObj as TreeViewComponent).addNodes(args.data);
        args.cancel = true;
    }
};
onTreeDragStop(args: DragAndDropEventArgs) {
    let kanbanElement: Element = <Element>closest(args.event.target as
Element, '#Kanban');
    if (kanbanElement) {
        let treeData: { [key: string]: Object }[] =
        (this.treeObj as TreeViewComponent).fields.dataSource as { [key:
string]: Object }[];
        const filteredData: { [key: string]: Object }[] =
        treeData.filter((item: any) => item.Id ===
parseInt(args.draggedNodeData['id'] as string, 10));
        (this.treeObj as
TreeViewComponent).removeNodes([args.draggedNodeData['id']] as string[]);
        (this.kanbanObj as KanbanComponent).openDialog('Add',
filteredData[0]);
        args.cancel = true;
    }
};
}

```

MAIN.TS

```

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

```

Schedule to Kanban

Drag the card from the Kanban board and drop it to the Schedule component and vice versa.

In the following sample, remove the data from the Kanban board using the `deleteCard` public method and add to the schedule component using the `addNodes` public method at Kanban `dragStop` event when dragging the card and dropping it to the Treeview component. Remove the data from Treeview using the `removeNodes` public method and add to Kanban board using the `addCard` public method when dragging the list from the Treeview component and dropping it to the kanban board.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { ScheduleAllModule, RecurrenceEditorAllModule } from
'@syncfusion/ej2-angular-schedule'
import { Component, ViewChild } from '@angular/core';
import { closest, removeClass, extend } from '@syncfusion/ej2-base';
import { CardSettingsModel, DragEventArgs, KanbanComponent } from
'@syncfusion/ej2-angular-kanban';
import { kanbanData, scheduleData } from './datasource';
import {
    EventSettingsModel, View, GroupModel, TimelineViewsService,
    TimelineMonthService,

```

```

    ResizeService, WorkHoursModel, DragAndDropService, ResourceDetails,
    ScheduleComponent, ResourcesModel
} from '@syncfusion/ej2-angular-schedule';
@Component({
  imports: [

    KanbanModule,
    ScheduleAllModule,
    RecurrenceEditorAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="container-fluid">
    <div class="row">
      <div class="col-md-4" style="width: 30%">
        <h4>Kanban</h4>
        <ejs-kanban id='Kanban' #Kanban keyField='DepartmentName'
[dataSource]='data' [externalDropId]='externalKanbanDropId'
[cardSettings]='cardSettings'
(dragStop)='onKanbanDragStop($event)'>
          <e-columns>
            <e-column headerText='GENERAL' keyField='GENERAL'></e-column>
          </e-columns>
        </ejs-kanban>
      </div>
      <div class="col-md-8" style="width: 70%">
        <h4>Schedule</h4>
        <ejs-schedule id='Schedule' #Schedule width='100%' height='650px'
[group]="group" [currentView]="currentView" [selectedDate]="selectedDate"
[workHours]="workHours" [eventSettings]="eventSettings"
(dragStop)="scheduleDragStop($event)">
          <e-resources>
            <e-resource field='DepartmentID' title='Department'
name='Departments' [dataSource]='departmentDataSource' textField='Text'
idField='Id' colorField='Color'>
              </e-resource>
            <e-resource field='ConsultantID' title='Consultant'
name='Consultants' [dataSource]='consultantDataSource'
[allowMultiple]='allowMultiple' textField='Text' idField='Id'
groupIDField="GroupId" colorField='Color'>
              </e-resource>
            </e-resources>
            <ng-template #resourceHeaderTemplate let-data>
              <div class="template-wrap">
                <div class="specialist-category">
                  <div class="specialist-
name">{{getConsultantName(data)}}</div>
                  <div class="specialist-
designation">{{getConsultantDesignation(data)}}</div>
                </div>
              </div>
            </ng-template>
            <e-views>
              <e-view option='TimelineDay'></e-view>
              <e-view option='TimelineMonth'></e-view>
            </e-views>
          </ejs-schedule>
        </div>
      </div>
    </div>`
})

```

```

        </div>
    </div>
</div>`,
    providers: [TimelineViewsService, TimelineMonthService, ResizeService,
DragAndDropService]
})
export class AppComponent {
    @ViewChild('Kanban')
    public kanbanObj?: KanbanComponent;
    @ViewChild('Schedule')
    public scheduleObj?: ScheduleComponent;
    public data: Object[] = kanbanData;
    public scheduleDataSource: Object[] = <Object[]>extend([], scheduleData,
undefined, true);
    public cardSettings: CardSettingsModel = {
        contentField: 'Name',
        headerField: 'Id'
    };
    public externalKanbanDropId: string[] = ['#Schedule'];
    public selectedDate: Date = new Date(2018, 7, 1);
    public currentView: View = 'TimelineDay';
    public workHours: WorkHoursModel = { start: '08:00', end: '18:00' };
    public departmentDataSource: Object[] = [
        { Text: 'GENERAL', Id: 1, Color: '#bbdc00' },
        { Text: 'DENTAL', Id: 2, Color: '#9e5fff' }
    ];
    public consultantDataSource: Object[] = [
        { Text: 'Alice', Id: 1, GroupId: 1, Color: '#bbdc00', Designation:
'Cardiologist' },
        { Text: 'Nancy', Id: 2, GroupId: 2, Color: '#9e5fff', Designation:
'Orthodontist' },
        { Text: 'Robert', Id: 3, GroupId: 1, Color: '#bbdc00', Designation:
'Optometrist' },
        { Text: 'Robson', Id: 4, GroupId: 2, Color: '#9e5fff', Designation:
'Periodontist' },
        { Text: 'Laura', Id: 5, GroupId: 1, Color: '#bbdc00', Designation:
'Orthopedic' },
        { Text: 'Margaret', Id: 6, GroupId: 2, Color: '#9e5fff',
Designation: 'Endodontist' }
    ];
    public group: GroupModel = { enableCompactView: false, resources:
['Departments', 'Consultants'] };
    public allowMultiple: Boolean = false;
    public eventSettings: EventSettingsModel = {
        dataSource: this.scheduleDataSource,
        fields: {
            subject: { title: 'Patient Name', name: 'Name' },
            startTime: { title: 'From', name: 'StartTime' },
            endTime: { title: 'To', name: 'EndTime' },
            description: { title: 'Reason', name: 'Description' }
        }
    };
    getConsultantName(value: ResourceDetails): string {
        return (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as ResourcesModel).textField as string] as string;
    }
    getConsultantStatus(value: ResourceDetails): boolean {

```

```

        let resourceName: string =
            (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as ResourcesModel).textField as string] as string;
            if (resourceName === 'GENERAL' || resourceName === 'DENTAL') {
                return false;
            } else {
                return true;
            }
        }
        getConsultantDesignation(value: ResourceDetails): string {
            let resourceName: string =
                (value as ResourceDetails).resourceData[((value as
ResourceDetails).resource as ResourcesModel).textField as string] as string;
            if (resourceName === 'GENERAL' || resourceName === 'DENTAL') {
                return '';
            } else {
                return (value as ResourceDetails).resourceData['Designation'] as
string;
            }
        }
        getConsultantImageName(value: ResourceDetails): string {
            return this.getConsultantName(value).toLowerCase();
        }
        onKanbanDragStop(args: DragEventArgs) {
            let scheduleElement: Element = <Element>closest(args.event.target as
Element, '#Schedule');
            if (scheduleElement) {
                (this.kanbanObj as KanbanComponent).deleteCard(args.data);
                (this.kanbanObj as any).openEditor(args.data[0], 'Add', true);
                args.cancel = true;
            }
        };
        scheduleDragStop(args: any) {
            let kanbanElement: Element = <Element>closest(args.event.target as
Element, '#Kanban');
            if (kanbanElement) {
                (this.scheduleObj as
ScheduleComponent).deleteEvent(args.data.Id);
                removeClass([(this.scheduleObj as
ScheduleComponent).element.querySelector('.e-selected-cell') as Element],
'e-selected-cell');
                this.kanbanObj?.openDialog('Add', args.data);
                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));

```

Sort in Angular Kanban component

The Kanban provides built-in support to arrange the cards in their columns based on the JSON data order and drop the cards in the columns based on the dropped clone. Initially, users can change the arrangement of cards in the columns and position of the dropped card by using the [sortBy](#) property. The [sortBy](#) property contains three enumeration values as follows.

- Index
- DataSourceOrder
- Custom

Index

SortBy **Index** property can be used with or without [field](#) mapping.

Index without field mapping

By default, SortBy **Index** property support without any [field](#) mapping. In this behavior, cards are loaded based on the JSON data order and cards are dropped based on the dropped clone.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}
```

MAIN.TS


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

Index with field mapping

SortBy **Index** property also supports with **field** mapping. In this behavior, cards are loaded based on mapping **field** values, and cards are dropped based on the dropped clone.

Cards are placed in a particular position in the columns where you can drop the cards by specifying the **field** property, which is mapped from the data source. This property allows the users to drop the cards in the Kanban board where the dropped clone is created exactly. It is also helpful to render the cards based on the **field** property value.

The **field** property mapping key value must be in **number** format.

The following cases will dynamically change their **field** value when dropping the cards.

- If the cell has no cards, the dropped card **field** value does not change.
- If the cell has one card and dropped a card to the last position or previous/next cards that do not have continuous order, then the dropped card **field** value will be changed based on their previous card value.
- If the cell has one card and dropped a card on the previous position, then it will compare both the values, and the dropped card **field** value will be changed if the cards have continuous order otherwise values will not be changed.
- When the previous and next cards do not have continuous order, the dropped card **field** value will be changed based on the previous card value.
- When the previous and next cards have continuous order or odd/even value, then the **field** value of the dropped card and the cards followed by the dropped card will be changed based on the **previous** card value with continuous order.

For Example,

Continuous Order -

Consider, Column A has Card A with priority value **1**, Card B with priority value **2**, and Card C with priority value **3**. and Column B has Card D with priority value **5**, then the dropped Card D will be placed between Card A and Card B. Now, the Cards D, B, and C will be dynamically changed to the priority values as **2**, **3**, and **4** respectively.

Odd/Even order -

Consider, Column A has Card A with priority value **1**, Card B with priority value **3**, and Card C with priority value **5**. and Column B has Card D with priority value **5**, then the Dropped Card D will be placed between Card A and Card B. Now, the Cards D, B, and C will be dynamically changed to the priority values as **2**, **3**, and **5** respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
```

```

import { Component } from '@angular/core';
import { CardSettingsModel, SortSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data' [cardSettings]='cardSettings' [sortSettings]='sortSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public sortSettings: SortSettingsModel = {
    sortBy: 'Index',
    field: 'RankId'
  };
}

```

MAIN.TS

```

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

```

DataSource Order

The `SortBy DataSourceOrder` property does not require any [field](#) mapping. In this behavior, cards are loaded based on the JSON data order, and also cards are dropped based on the JSON data order.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, SortSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';

```

```

@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [sortSettings]='sortSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public sortSettings: SortSettingsModel = {
    sortBy: 'DataSourceOrder'
  };
}

```

MAIN.TS

```

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

```

Custom

Custom with field mapping

The SortBy **Custom** property must require datasource [field](#) mapping. In this behavior, cards are loaded based on the [field](#) mapping value and also cards are dropped based on the [field](#) mapping value.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SortSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule

```

```

    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [sortSettings]='sortSettings'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
  })
  export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
      contentField: 'Summary',
      headerField: 'Id'
    };
    public sortSettings: SortSettingsModel = {
      sortBy: 'Custom',
      field: 'Summary'
    };
  }
}

```

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 direction

Kanban board also provides support for aligning the cards in the columns using the [direction](#) property inside the [sortSettings](#) property. Based on this, cards can be aligned in the columns either in **Ascending** or **Descending** order. Sorting direction will be performed based on [sortBy](#) property.

By default, cards are aligned in the columns based on **Ascending** order.

In the following sample, cards are aligned in **Descending** order.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SortSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

```

```

        KanbanModule
      ],
      standalone: true,
      selector: 'app-root',
      template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [sortSettings]='sortSettings'>
        <e-columns>
          <e-column headerText='To do' keyField='Open'></e-column>
          <e-column headerText='In Progress'
keyField='InProgress'></e-column>
          <e-column headerText='Testing' keyField='Testing'></e-
column>
          <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
      </ejs-kanban>`
    })
    export class AppComponent {
      public data: Object[] = kanbanData;
      public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id'
      };
      public sortSettings: SortSettingsModel = {
        sortBy: 'Custom',
        field: 'Summary',
        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));

```

Dialog in Angular Kanban component

The Kanban provides built-in support to add, edit and delete a card using dialog module. User can edit a card using the following ways.

- Built-in dialog module
- Custom Fields
- Dialog template

Default Dialog

When double-click on the cards, the dialog is opened with below fields to edit a card. This dialog contains **Delete**, **Save** and **Cancel** buttons.

- To edit a card, modify the card details and click the **Save** button.
- To delete a card, click **Delete** button.
- Click on the **Cancel** button to cancel the editing action.

The dialog displays with the following fields which mapped to dialog fields by default.

Key | Type | Text

cardSettings.headerField | Input | ID

keyField | DropDown | -

cardSettings.contentField | TextArea | -

cardSettings.priority(If applicable) | Numeric | -

swimlaneSettings.keyField(If applicable) | DropDown | -

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}
```

MAIN.TS

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

Custom Fields

You can change the default fields of dialog using `fields` property inside the `dialogSettings` property. The `key` property used to map the `dataSource` value and rendered the corresponding component based on specified `type` property.

The following types are available in dialog fields.

- String
- Numeric
- TextArea
- DropDown
- TextBox
- Input

If `type` is not defined in the fields, then it renders as the HTML input element in dialog.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [dialogSettings]='dialogSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public dialogSettings: DialogSettingsModel = {
    fields: [
      { key: 'Id', type: 'Input' },
      { key: 'Status', type: 'DropDown' },
      { key: 'Estimate', type: 'Numeric' },
```

```

        { key: 'Summary', type: 'TextArea' }
      ]
    } as DialogSettingsModel;
  }

```

MAIN.TS

```

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

```

Custom Fields label

By default, the fields **key** mapping value is considered as a **label** and you can change this label by using **text** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, DialogSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [dialogSettings]='dialogSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public dialogSettings: DialogSettingsModel = {
    fields: [
      { text: 'ID', key: 'Id', type: 'Input' },
      { key: 'Status', type: 'DropDown' },
      { key: 'Estimate', type: 'Numeric' },

```



```

        { key: 'Summary', type: 'TextArea' }
      ]
    } as DialogSettingsModel;
  }

```

MAIN.TS

```

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

```

Fields Validation

The dialog fields can be validated while click on the **Save** button. This can be achieved by using **validationRules** property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel, DialogSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [dialogSettings]='dialogSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public dialogSettings: DialogSettingsModel = {
    fields: [
      { text: 'ID', key: 'Id', type: 'Input' },
      { key: 'Status', type: 'DropDown' },

```

```

        { key: 'Estimate', type: 'Numeric', validationRules: { range:
[0, 1000] } },
        { key: 'Summary', type: 'TextArea', validationRules: { required:
true } }
    ]
} as DialogSettingsModel;
}

```

MAIN.TS

```

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

```

Dialog Template

Using the dialog template, you can render your own dialog by defining the `template` property. Initialize the template as SCRIPT element Id or HTML string which holds the template and map it to the template property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { DropDownListAllModule } from '@syncfusion/ej2-angular-dropdowns';
import { NumericTextBoxAllModule, TextBoxAllModule } from '@syncfusion/ej2-angular-inputs';
import { Component } from '@angular/core';
import { DropDownList } from '@syncfusion/ej2-dropdowns';
import { NumericTextBox, TextBox } from '@syncfusion/ej2-inputs';
import { CardSettingsModel, DialogSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule,
    DropDownListAllModule,
    NumericTextBoxAllModule,
    TextBoxAllModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
    <ng-template #dialogSettingsTemplate let-data>

```

```

        <table>
            <tbody>
                <tr>
                    <td class="e-label">ID</td>
                    <td>
                        <div class="e-float-input e-control-
wrapper">
                            <input id="Id" name="Id" type="text"
class="e-field" value='{{data.Id}}' disabled />
                        </div>
                    </td>
                </tr>
                <tr>
                    <td class="e-label">Status</td>
                    <td>
                        <ejs-dropdownlist id='Status'
#dropdownStatus name="Status"
                            class="e-field"
[dataSource]='statusData' value='{{data.Status}}'
                            placeholder='Status'>
                        </ejs-dropdownlist>
                    </td>
                </tr>
                <tr>
                    <td class="e-label">Assignee</td>
                    <td>
                        <ejs-dropdownlist id='Assignee'
#dropdownAssignee name="Assignee"
                            class="e-field"
[dataSource]='assigneeData'
                            value='{{data.Assignee}}'
placeholder='Assignee'></ejs-dropdownlist>
                    </td>
                </tr>
                <tr>
                    <td class="e-label">Priority</td>
                    <td>
                        <ejs-dropdownlist type="text"
name="Priority" id="Priority"
                            class="e-field"
placeholder='Priority' [dataSource]='priorityData'
                            value='{{data.Priority}}'></ejs-
dropdownlist>
                    </td>
                </tr>
                <tr>
                    <td class="e-label">Summary</td>
                    <td>
                        <div class="e-float-input e-control-
wrapper">
                            <textarea type="text" name="Summary"
id="Summary" class="e-field"
                            value='{{data.Summary}}'></textarea>
                        </div>
                    </td>
                </tr>
            </tbody>
        </table>

```

```

        </table>
      </ng-template>
    </ejs-kanban>`
  })
  export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
      contentField: 'Summary',
      headerField: 'Id'
    };
    statusData: any;
    assigneeData: any;
    priorityData: any;
  }
}

```

MAIN.TS

```

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

```

Prevent Dialog

The Kanban allows to prevent to open a dialog on card double-click by enabling `args.cancel` in `dialogOpen` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, DialogSettingsModel, DialogEventArgs } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' [dialogSettings]='dialogSettings'
(dialogOpen)='dialogOpen($event)'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})

```

```
export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
        contentField: 'Summary',
        headerField: 'Id'
    };
    dialogSettings: any;
    dialogOpen(args: DialogEventArgs): void {
        args.cancel = true;
    }
}
```

MAIN.TS

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

Persisting data in server

The modified card data can be persisted in the database using the RESTful web services. All the CRUD operations in the Kanban are done through [DataManager](#). The [DataManager](#) has an option to bind all the CRUD related data in server-side.

For your information, the ODataAdaptor persists data in the server as per OData protocol.

In the below section covers how to get the edited data details on the server-side using the [UrlAdaptor](#).

URL adaptor

You can use the [UrlAdaptor](#) of [DataManager](#) when binding datasource for remote data. In the initial load of Kanban, data are fetched from remote data and bound to the Kanban using `url` property of [DataManager](#).

You can map the CRUD operation in Kanban can be mapped to server-side controller actions using the properties `insertUrl`, `removeUrl`, `updateUrl`, and `crudUrl`.

- `insertUrl` – You can perform single insertion operation on server-side.
- `updateUrl` – You can update single data on server-side.
- `removeUrl` – You can remove single data on server-side.
- `crudUrl` – You can perform bulk data operation on server-side.

The following code example describes the above behavior.

`typescript

```
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
@Component({
    selector: 'app-root',
```

```

template: `<ejs-kanban keyField='Status' [dataSource]='dataManager' [cardSettings]='cardSettings'>
<e-columns>
<e-column headerText='To do' keyField='Open'></e-column>
<e-column headerText='In Progress' keyField='InProgress'></e-column>
<e-column headerText='Testing' keyField='Testing'></e-column>
<e-column headerText='Done' keyField='Close'></e-column>
</e-columns>
</ejs-kanban>`
})
export class AppComponent {
public cardSettings: CardSettingsModel = {
contentField: 'Summary',
headerField: 'Id'
};
private dataManager: DataManager = new DataManager({
url: 'Home/DataSource',
updateUrl: 'Home/Update',
insertUrl: 'Home/Insert',
removeUrl: 'Home/Delete',
adaptor: new UrlAdaptor(),
crossDomain: true
});
}
`

```

The server-side controller code to handle the CRUD operations are as follows.

```

`typescript
private NORTHWNDEntities db = new NORTHWNDEntities();
public ActionResult DataSource() {
var DataSource = db.Tasks.ToList();
return Json(DataSource, JsonRequestBehavior.AllowGet);
}
public ActionResult Insert(Params value) {
//Insert card data into the database

```

```

return Json(value, JsonRequestBehavior.AllowGet);
}
public ActionResult Update(Params value) {
//Update card data into the database
return Json(value, JsonRequestBehavior.AllowGet);
}
public void Delete(Params value) {
//Delete card data from the database
}
public class Params {
public int Id { get; set; }
public string Status { get; set; }
public string Summary { get; set; }
public string Assignee { get; set; }
}
`

```

Insert card

Using the `insertUrl` property, you can specify the controller action mapping URL to perform insert operation on the server-side.

The following code example describes the above behavior.

```

`typescript
public ActionResult Insert(Params value)
{
//Insert card in the database
}
`

```

The newly added record details are bound to the `value` parameter.

Update card

Using the `updateUrl` property, the controller action mapping URL can be specified to perform save/update operation on the server-side.

The following code example describes the above behavior.

```

`typescript
public ActionResult Update(Params value)
{

```

```
//Update card data in the database
}
```

The updated record details are bound to the `value` parameter.

Delete card

Using the `removeUrl` property, the controller action mapping URL can be specified to perform delete operation on the server-side.

The following code example describes the above behavior.

```
`typescript
public void Delete(int key)
{
//Delete card in the database
}
```

The deleted card primary key value is bound to the `key` parameter.

CRUD URL

Using the `crudUrl` property, the controller action mapping URL can be specified to perform all the CRUD operations at the server-side using a single method instead of specifying a separate controller action method for CRUD (insert, update and delete) operations.

The action parameter of `crudUrl` is used to get the corresponding CRUD action.

The following code example describes the above behavior.

```
`typescript
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { DataManager, UrlAdaptor } from '@syncfusion/ej2-data';
@Component({
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='dataManager' [cardSettings]='cardSettings'>
<e-columns>
<e-column headerText='To do' keyField='Open'></e-column>
<e-column headerText='In Progress' keyField='InProgress'></e-column>
<e-column headerText='Testing' keyField='Testing'></e-column>
<e-column headerText='Done' keyField='Close'></e-column>
</e-columns>`
```



```

</ejs-kanban>`
})
export class AppComponent {
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  private dataManager: DataManager = new DataManager({
    url: 'Home/DataSource',
    updateUrl: 'Home/UpdateData',
    insertUrl: 'Home/UpdateData',
    removeUrl: 'Home/UpdateData',
    crudUrl: 'Home/UpdateData',
    adaptor: new UrlAdaptor(),
    crossDomain: true
  });
}
`typescript
private NORTHWNDEntities db = new NORTHWNDEntities();
public ActionResult DataSource() {
  var DataSource = db.Tasks.ToList();
  return Json(DataSource, JsonRequestBehavior.AllowGet);
}
public ActionResult UpdateData(EditParams param) {
  if (param.action == "insert" || (param.action == "batch" && param.added != null)) {
    if (param.action == "insert") {
      db.Tasks.Add(param.value);
    } else {
      foreach (var temp in param.added) {
        db.Tasks.Add(temp);
      }
    }
  }
}

```

```
}  
if (param.action == "update" || (param.action == "batch" && param.changed != null)) {  
    if (param.action == "update") {  
        Task old = db.Tasks.Where(o => o.Id == param.value.Id).SingleOrDefault();  
        if (old != null) {  
            db.Entry(old).CurrentValues.SetValues(param.value);  
        }  
    } else {  
        foreach (var temp in param.changed) {  
            Task old = db.Tasks.Where(o => o.Id == temp.Id).SingleOrDefault();  
            if (old != null) {  
                db.Entry(old).CurrentValues.SetValues(temp);  
            }  
        }  
    }  
}  
if (param.action == "remove" || (param.action == "batch" && param.deleted != null)) {  
    if (param.action == "remove") {  
        int key = Convert.ToInt32(param.key);  
        db.Tasks.Remove(db.Tasks.Where(o => o.Id == key).SingleOrDefault());  
    } else {  
        foreach (var temp in param.deleted) {  
            db.Tasks.Remove(db.Tasks.Where(o => o.Id == temp.Id).SingleOrDefault());  
        }  
    }  
}  
db.SaveChanges();  
return Json(param, JsonRequestBehavior.AllowGet);  
}  
public class EditParams {  
    public string key { get; set; }  
    public string action { get; set; }  
    public List<Tasks> added { get; set; }  
}
```

```

public List<Tasks> changed { get; set; }
public List<Tasks> deleted { get; set; }
public Tasks value { get; set; }
}
`

```

The `crudUrl` is used to update the bulk data sent to the server-side. Multiple selections and `sortBy` as `Index` properties are used for `crudUrl` properties to update the modified bulk data to the server-side.

Tooltip in Angular Kanban component

The tooltip is used to show the card information when the cursor hover over the card elements using the `enableTooltip` property. Tooltip content is dynamically set based on hovering over the card elements.

If you wish to show tooltip on Kanban board custom elements, you need to add `e-tooltip-text` class name of a particular element.

Tooltip template

You can customize the tooltip content with any HTML or CSS element and styling using the `tooltipTemplate` property. In the following demo, the tooltip is customized with HTML elements.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' enableTooltip='true'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
    <ng-template #tooltipTemplate let-data>
      <div class='e-kanbanTooltipTemp'>
        <table>
          <tr>
            <td class="details">
              <table>

```

```

class="CardHeader">Assignee:</td>
class="CardHeader">Type:</td>
class="CardHeader">Estimate:</td>
class="CardHeader">Summary:</td>
    <colgroup>
      <col style="width:30%">
      <col style="width:70%">
    </colgroup>
    <tbody>
      <tr>
        <td>
          <td>{{data.Assignee}}</td>
        </tr>
      <tr>
        <td>
          <td>{{data.Type}}</td>
        </tr>
      <tr>
        <td>
          <td>{{data.Estimate}}</td>
        </tr>
      <tr>
        <td>
          <td>{{data.Summary}}</td>
        </tr>
      </tbody>
    </table>
  </td>
</tr>
</table>
</div>
</ng-template>
</ejs-kanban>`
  })
  export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
      contentField: 'Summary',
      headerField: 'Id'
    };
  }
}

```

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 Kanban component

Validate particular column using the `minCount` or `maxCount` properties. The corresponding columns gets different appearance when validation fails. In default layout, `constraintType` property accept only `Column` type. In swimlane layout, accept both `Column` and `Swimlane` constraint type.

There are two types of constraints:

1. Column
2. Swimlane

By default, the column count validation is performed based on Kanban **columns**.

Minimum card limit

The **minCount** property is used to specify the minimum cards hold on particular column or swimlane cell. If the column or swimlane total card count falls short of the minimum count value, it shows the column or cell background colour with validation fails.

Maximum card limit

The **maxCount** property is used to specify the maximum cards hold on particular column or swimlane cell. If the column or swimlane cell total card count exceeds the maximum count value, it shows the column or cell background colour with validation fails.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'
minCount='6'></e-column>
      <e-column headerText='In Progress' keyField='InProgress'
maxCount='5'></e-column>
      <e-column headerText='Testing' keyField='Testing'
minCount='3' maxCount='4'></e-column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}
```

MAIN.TS

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

Virtualization in Angular Kanban component

Kanban allows you to load a large amount of data without any performance degradation. This feature can be enabled by setting the [enableVirtualization](#) property in the Kanban to **true**.

Virtual scrolling

Virtual scrolling optimizes data rendering within each column when using large datasets. Only a subset of cards that are visible and about to be loaded on the screen are rendered. The number of records displayed in the Kanban is determined implicitly by the height of the Kanban area and the card height. The [cardHeight](#) property of Kanban can be used to set the cards' height in pixel value. By default, the card height will be **auto**.

When the Kanban column is scrolled, the virtual scrolling feature dynamically loads additional data on demand into view and unloads the data that is no longer visible.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { ColumnsModel, CardSettingsModel, DialogSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { generateKanbanDataVirtualScrollData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban #kanbanObj id='KanbanVirtualScrolling'
enableVirtualization='true'
[dataSource]='kanbanData' keyField='Status' enableTooltip='true'
[columns]='columnsSettings' [cardSettings]='cardSettings'
[dialogSettings]='dialogSettings'>
</ejs-kanban>`
})
export class AppComponent {
  public kanbanData: Object[];
  constructor() {
    this.kanbanData = generateKanbanDataVirtualScrollData();
  }
  public columnsSettings: ColumnsModel[] = [
    { headerText: 'To Do', keyField: 'Open' },
    { headerText: 'In Progress', keyField: 'InProgress' },
    { headerText: 'Code Review', keyField: 'Review' },
    { headerText: 'Done', keyField: 'Close' }
  ];
  public cardSettings: CardSettingsModel = {
    headerField: 'Id',
```

```

        contentField: 'Summary',
        selectionType: 'Multiple'
    };
    public dialogSettings: DialogSettingsModel = {
        fields: [
            { key: 'Id', text: 'ID', type: 'TextBox' },
            { key: 'Status', text: 'Status', type: 'DropDown' },
            { key: 'StoryPoints', text: 'Story Points', type: 'Numeric' },
            { key: 'Summary', text: 'Summary', type: 'TextArea' }
        ]
    };
}

```

MAIN.TS

```

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

```

Configure the remote data service

When the remote data is configured for the [dataSource](#), the service method will receive an additional **KanbanVirtualization** parameter to handle the initial data load for Kanban Virtualization.

To handle Kanban virtual scrolling, the server-side code needs to handle the **Where** and **Take** queries differently using the **KanbanVirtualization** parameter. The following is the example code for handling Kanban virtualization's initial data load using the **KanbanVirtualization** parameter.

```

`ts
public IActionResult LoadCard([FromBody] ExtendedDataManagerRequest dm)
{
    kanbanData = _context.KanbanCards.ToList();
    IEnumerable<KanbanCard> DataSource = kanbanData.AsEnumerable();
    DataOperations operation = new DataOperations();
    // For normal kanban data load Where query handling.
    if (dm.Where != null && dm.Where.Count > 0 && dm.KanbanVirtualization != "KanbanVirtualization")
    {
        dm.Where[0].value = dm.Where[0].value.ToString();
        DataSource = operation.PerformFiltering(DataSource, dm.Where, dm.Where[0].Operator);
    }
    if (dm.Skip != 0)
    {
        DataSource = operation.PerformSkip(DataSource, dm.Skip);
    }
}

```

```
// For normal Kanban data load Take query handling.
if (dm.Take != 0 && dm.KanbanVirtualization != "KanbanVirtualization")
{
    DataSource = operation.PerformTake(DataSource, dm.Take);
}

// For Kanban virtual scrolling data load Where and Take query handling.
var columnCount = new List<KeyValuePair<string, int>>();
if (dm.KanbanVirtualization == "KanbanVirtualization" && dm.Where != null && dm.Where.Count > 0
    && dm.Take != 0)
{
    IEnumerable<KanbanCard> currentData = new List<KanbanCard>();
    List<WhereFilter> currentFilter = new List<WhereFilter>();
    for (int i = 0; i < dm.Where.Count; i++)
    {
        dm.Where[i].value = dm.Where[i].value.ToString();
        currentFilter.Add(dm.Where[i]);
        var filterData = operation.PerformFiltering(DataSource, currentFilter, dm.Where[i].Operator);
        columnCount.Add(new KeyValuePair<string, int>(dm.Where[i].value.ToString(), filterData.Count()));
        filterData = operation.PerformTake(filterData, dm.Take);
        currentData = currentData.Concat(filterData);
        currentFilter.Clear();
    }
    DataSource = currentData;
}

// To return the data for Kanban virtual scrolling.
if (dm.KanbanVirtualization == "KanbanVirtualization") {
    return Json(new { result = DataSource, count = columnCount });
}

// To return the data for Kanban virtual scrolling.
else
{
    return Json(DataSource);
}
```



```
}
,
```

Limitations for virtual scrolling

- When virtualization is enabled in a Kanban board and the card height is not explicitly set, it will not default to `auto` height. Instead, a fixed height of `100px` will be applied to the cards. It's important to note that the card height should be specified in pixel values, as percentage values are not accepted.
- When a card is dragged and dropped, the index position of the card will not be preserved when scrolling through the column.
- Virtualization is not supported for swimlanes in the Kanban board.

Localization in Angular Kanban component

The localization library allows you to localize the default text content of the Kanban to different cultures using the `locale` property.

In Kanban, total count and min or max count text alone will be localized based on culture.

```
| Locale key | en-US (default) |
|-----|-----|
| items | items |
| min | Min |
| max | Max |
| cardsSelected | Cards Selected |
| addTitle | Add New Card |
| editTitle | Edit Card Details |
| deleteTitle | Delete Card |
| deleteContent | Are you sure you want to delete this card? |
| save | Save |
| delete | Delete |
| cancel | Cancel |
| yes | Yes |
| no | No |
| close | Close |
| noCard | No cards to display |
| unassigned | Unassigned |
```

Loading translations

To load translation object in an application, use `load` function of `L10n` class.

The following example demonstrates the Kanban in `Deutsch` culture.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-
angular-kanban';
import { kanbanData } from './datasource';
L10n.load({
  'de': {
    'kanban': {
      'items': 'Artikel',
      'min': 'Min',
      'max': 'Max',
      'cardsSelected': 'Karten ausgewählt',
      'addTitle': 'Neue Karte hinzufügen',
      'editTitle': 'Kartendetails bearbeiten',
      'deleteTitle': 'Karte löschen',
      'deleteContent': 'Möchten Sie diese Karte wirklich löschen?',
      'save': 'speichern',
      'delete': 'Löschen',
      'cancel': 'Stornieren',
      'yes': 'Ja',
      'no': 'Nein',
      'close': 'Schließen',
      'noCard': 'Keine Karten zum Anzeigen',
      'unassigned': 'nicht zugewiesen'
    }
  }
});
@Component({
  imports: [

    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' locale='de'
[swimlaneSettings]='swimlaneSettings'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'
minCount='6'></e-column>
      <e-column headerText='In Progress' keyField='InProgress'
maxCount='3'></e-column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
};

```

```

    public swimlaneSettings: SwimlaneSettingsModel = {
        keyField: 'Assignee'
    };
}

```

MAIN.TS

```

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

```

Right to left (RTL)

The Kanban provides an option to switch its text direction and layout from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc.). To enable right-to-left mode in Kanban, set the `enableRtl` to true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
L10n.load({
    'ar': {
        'kanban': {
            'items': 'العناصر',
            'min': 'أنا',
            'max': 'ماكس',
            'cardsSelected': 'تم تحديد البطاقات',
            'addTitle': 'إضافة بطاقة جديدة',
            'editTitle': 'تحرير تفاصيل البطاقة',
            'deleteTitle': 'حذف البطاقة',
            'deleteContent': 'هل أنت متأكد أنك تريد حذف هذه البطاقة؟',
            'save': 'حفظ',
            'delete': 'حذف',
            'cancel': 'إلغاء',
            'yes': 'نعم',
            'no': 'لا',
            'close': 'قريب',
            'noCard': 'لا توجد بطاقات لعرضها',
            'unassigned': 'غير معين'
        }
    }
});
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,

```

```

    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' locale='ar'
[swimlaneSettings]='swimlaneSettings' enableRtl='true'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'
minCount='6'></e-column>
            <e-column headerText='In Progress' keyField='InProgress'
maxCount='3'></e-column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
  })
  export class AppComponent {
    public data: Object[] = kanbanData;
    public cardSettings: CardSettingsModel = {
      contentField: 'Summary',
      headerField: 'Id'
    };
    public swimlaneSettings: SwimlaneSettingsModel = {
      keyField: 'Assignee'
    };
  }
}

```

MAIN.TS

```

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

```

Dimensions in Angular Kanban component

The Kanban dimensions refers to both height and width of the entire layout and it accepts three types of values.

- Auto
- Pixel
- Percentage

Auto height and width

When height and width of the Kanban are set to **auto**, it will try as hard as possible to keep an element the same width as its parent container. In other words, the parent container that holds Kanban, its width or height will be the sum of its children. By default, Kanban is assigned with **auto** values for both the height and width properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';

```

```

@Component({
  imports: [

    KanbanModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' width='auto' height='auto'>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
}

```

MAIN.TS

```

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

```

Height and width in pixel

The Kanban height and width will be rendered exactly as per the given pixel values. It accepts both string and number values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' width=650 height='550px'>

```

```

        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
    }
}

```

MAIN.TS

```

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

```

Height and width in percentage

When height and width of the Kanban are given in percentage, it will make the Kanban as wide as the parent container.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { KanbanModule } from '@syncfusion/ej2-angular-kanban';
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [
        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' width='100%' height='100%'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'></e-column>
            <e-column headerText='In Progress'
keyField='InProgress'></e-column>
            <e-column headerText='Testing' keyField='Testing'></e-
column>
            <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
    </ejs-kanban>`
})

```

```

    })
    export class AppComponent {
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
    }
}

```

MAIN.TS

```

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

```

Persistence in Angular Kanban component

State persistence refers to the Kanban state maintained in the browser's [localStorage](#) even if the browser is refreshed or if you move to the next page within the browser.

State persistence stores Kanban datasource, column and swimlane expand/collapse state in the local storage when the [enablePersistence](#) is defined as true.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel, SwimlaneSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
    imports: [

        KanbanModule
    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-kanban keyField='Status' [dataSource]='data'
    [cardSettings]='cardSettings' [swimlaneSettings]='swimlaneSettings'
    enablePersistence='true'>
        <e-columns>
            <e-column headerText='To do' keyField='Open'
allowToggle='true'></e-column>
            <e-column headerText='In Progress' keyField='InProgress'
allowToggle='true'></e-column>
            <e-column headerText='Testing' keyField='Testing'
allowToggle='true'></e-column>
            <e-column headerText='Done' keyField='Close'
allowToggle='true'></e-column>
        </e-columns>
    </ejs-kanban>`
})
export class AppComponent {

```

```
public data: Object[] = kanbanData;
public cardSettings: CardSettingsModel = {
  contentField: 'Summary',
  headerField: 'Id'
};
public swimlaneSettings: SwimlaneSettingsModel = { keyField: 'Assignee'
};
}
```

MAIN.TS

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

Responsive mode in Angular Kanban component

The Kanban component has support for responsive behavior based on the client browser's width and height.

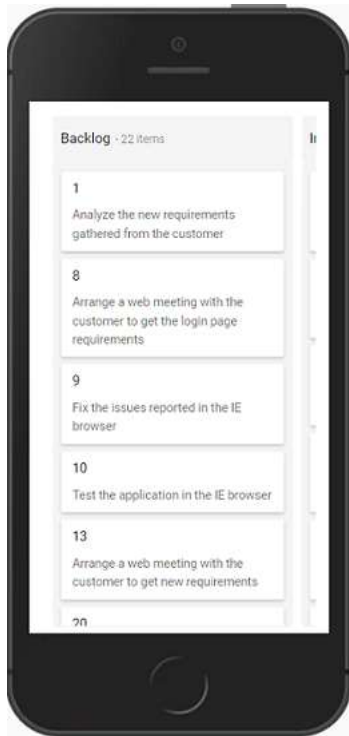
Layouts

Possible layouts are:

- Default Layout
- Swimlane Layout

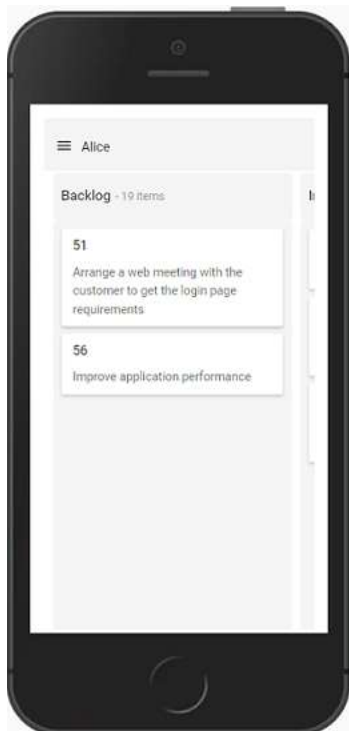
Default Layout

Kanban user interface is customized and redesigned for the best view on small screens. In responsive mode, the first column occupies 80% and the second column occupies 20% of the screen layout. Tap and hold the Kanban card to drag and drop it. Swipe left or right to view the columns.



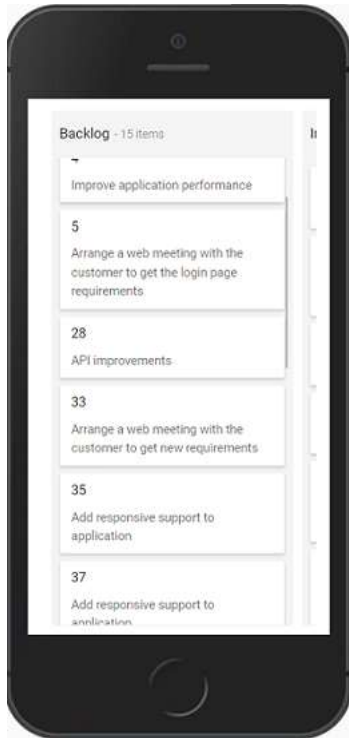
Swimlane Layout

Kanban swimlane header is rendered with menu icon on top of the kanban board. It will show all the available swimlane groups of the header text with a popup when clicking the menu icon. Swimlane selected grouped header text resultant data is rendered on the Kanban board. By default, the first swimlane grouped header text is selected and the resultant data is shown on the Kanban board. The Kanban board data will be changed when changing the swimlane group header text.



Scrolling

Column scrolling will be shown when exceeding the screen size in the columns.

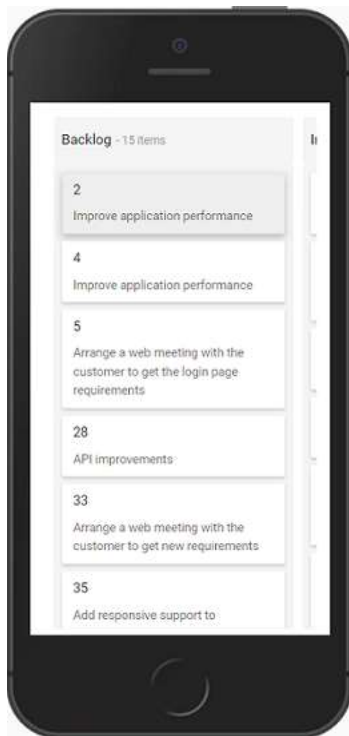


Selection

Select particular cards in the Kanban board by tapping the card.

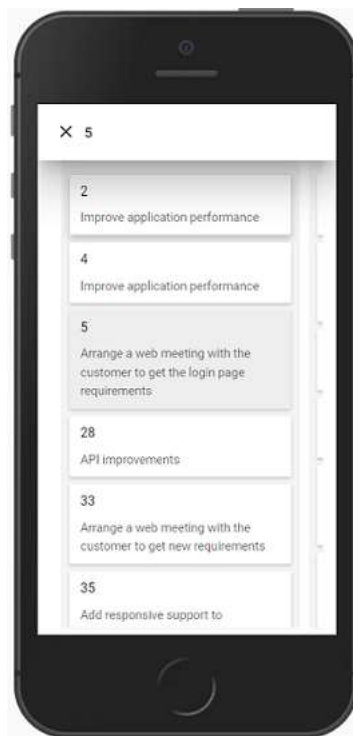
Single Selection

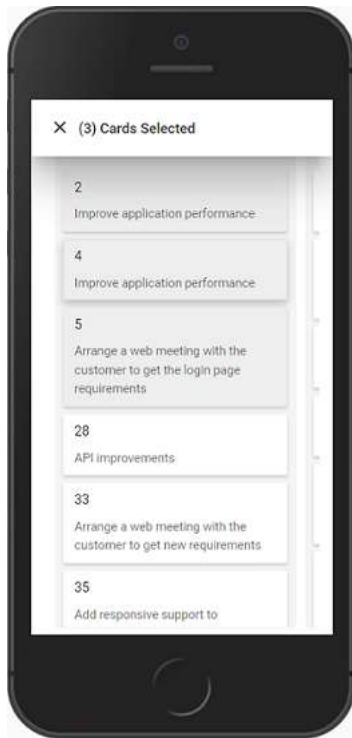
Single card will be selected when you tap the card once and selection will be removed when you select another card.



Multiple Selection

Enable [selectionType](#) as `Multiple` to select multiple cards. It will open the popup on the screen top. Selected card header text will be shown when selecting single card with a tap and hold action. If single card is selected, only tap action is required to select multiple cards. Multiple Selected card count will be shown on the popup when selecting multiple cards.





Style in Angular Kanban component

To modify the Kanban appearance, you need to override the default CSS of Kanban. Also, you have an option to create your own custom theme using our [Theme Studio](#). Please find the list of CSS classes in Kanban.

Class	Purpose
<code>.e-kanban</code>	Customize the kanban.
<code>.e-kanban .e-kanban-header .e-header-cells</code>	Header cells of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-header-wrap .e-header-title</code>	Header title of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-min-color</code>	Header cells minimum color of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-max-color</code>	Header cells maximum color of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-collapsed.e-min-color</code>	Header cells of collapsed column minimum color in column constraint type of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells.e-collapsed.e-max-color</code>	Header cells of collapsed column maximum color in column constraint type of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-header-text</code>	Header text of Kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-item-count</code>	Header cells Item count of Kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-limits</code>	Header cells limits in column constraint type of kanban.
<code>.e-kanban .e-kanban-header .e-header-cells .e-limits .e-min-count</code>	Header cells minimum count of kanban.

| .e-kanban .e-kanban-header .e-header-cells .e-limits .e-max-count | Header cells maximum count of kanban. |

| .e-kanban .e-kanban-content | Customize kanban Content. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits | Content cells limits in swimlane constraint type of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits .e-min-count | Content cells minimum count of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-limits .e-max-count | Content cells maximum count of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-min-color | Content cells minimum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-max-color | Content cells maximum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-collapsed .e-collapse-header-text | Content cells of collapsed header text. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells.e-collapsed .e-collapse-header-text .e-item-count | Content cells of collapsed header text Item count. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-show-add-button | Add button in content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-show-add-button .e-show-add-icon | Customize content cells add icon of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-empty-card | Empty content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card | Customize cards in kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-header .e-card-header-title | Cards header title of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-footer | Cards footer of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-content | Cards content of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card.e-card-color | Cards color of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-tags | Customize Card tags of kanban. |

| .e-kanban .e-kanban-content .e-content-row .e-content-cells .e-card-wrapper .e-card .e-card-tag | Card tag of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-row-expand | Content cells of swimlane row expand of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-row-collapse | Content cells of swimlane row collapse of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-swimlane-text | Content cells of swimlane header text of kanban. |

| .e-kanban .e-kanban-content .e-content-row.e-swimlane-row .e-content-cells .e-swimlane-header .e-item-count | Content cells of swimlane items count of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells | swimlane content cells of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-dropping | Customize swimlane content cells card dropping of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells .e-card-wrapper | Swimlane content cells of card wrapper. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-min-color | Swimlane content cells of minimum color of kanban. |

| .e-kanban .e-kanban-content .e-content-row:not(.e-swimlane-row) .e-content-cells.e-max-color | Swimlane content cells of maximum color of kanban. |Customize the kanban CSS theme. Please find the list of CSS classes in Kanban. | .e-kanban .e-kanban-table .e-header-cells | Header cells of kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-header-text | Header text of Kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-item-count | Header cells Item count of Kanban. |

| .e-kanban .e-kanban-table .e-header-cells .e-column-expand | Header cells of toggle icon in column expand. |

| .e-kanban .e-kanban-table .e-header-cells .e-column-collapse | Header cells of toggle icon in column collapse. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row:not(.e-swimlane-row) .e-content-cells | swimlane content cells of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row.e-swimlane-row .e-swimlane-text | Content cells of swimlane header text of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row.e-swimlane-row .e-item-count | Content cells of swimlane items count of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-content-row .e-show-add-button .e-show-add-icon | Add icon in content cells of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-card.e-selection | Selected card of kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-card .e-card-header | Cards header in kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-card .e-card-content | Cards content in kanban. |

| .e-kanban .e-kanban-table.e-content-table .e-card .e-card-tag.e-card-label | Cards label in kanban. |

[To set fixed position to the Kanban header](#)

The Fixed header in Kanban control can be customized in following ways,

By setting a fixed height to the Kanban content,

```
`CSS
```

```
.e-kanban .e-kanban-content {
height: 500px;
}
`
```

By customizing the CSS for the Kanban header.

```
`CSS
```

```
.e-kanban-header {
position: -webkit-sticky;
position: sticky;
z-index: 100;
top: 0;
}
`
```

Note: It will not affect the Kanban content's height.

Accessibility in Angular Kanban component

The Kanban component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties. This component is characterized by complete ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The accessibility compliance for the Kanban component is outlined below.

Accessibility Criteria	Compatibility
--	--
WCAG 2.2 Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Section 508 Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Screen Reader Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Right-To-Left Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Color Contrast	![Intermediate](https://cdn.syncfusion.com/content/images/documentation/partial.png)
Mobile Device Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Keyboard Navigation Support	![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png)
Accessibility Checker Validation	![Intermediate](https://cdn.syncfusion.com/content/images/documentation/partial.png)

| [Axe-core Accessibility Validation](#) |

![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png) |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

![Yes](https://cdn.syncfusion.com/content/images/documentation/full.png) - All features of the component meet the requirement.

![Intermediate](https://cdn.syncfusion.com/content/images/documentation/partial.png) - Some features of the component do not meet the requirement.

![No](https://cdn.syncfusion.com/content/images/documentation/not-supported.png) - The component does not meet the requirement.

WAI-ARIA attributes

The Kanban component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Kanban component:

| Attributes | Purpose |

| --- | --- |

| **aria-label** | It helps to provides information about elements in a kanban component for assistive technology. |

| **aria-expanded** | Attributes indicate the state of a collapsible element. |

| **aria-selected** | This attribute is assigned to the Kanban component for the selection of elements, and its default value is **false**. The value changes to true when the user selects a Kanban card. |

| **aria-grabbed** | Indicates whether the attribute is set to true. It has been selected for dragging. If this attribute is set to false, the element can be grabbed for a drag-and-drop operation but will not be currently grabbed. |

| **aria-describedby** | This attribute contains the ID of the Kanban header column to indicate that the attribute establishes an association between the Kanban header column and the Kanban column body. |

| **aria-roledescription** | This attribute is assigned to the Kanban component and is used to provide alternative descriptions for card elements. |

Keyboard interaction

The Kanban component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Kanban component.

| **Press** | **To do this** |

| --- | --- |

- | Home | To select the first card in the kanban |
- | End | To select the last card in the kanban |
- | Arrow Up | Select the card through the up arrow |
- | Arrow Down | Select the card through the down arrow |
- | Arrow Right | Move the column selection to the right |
- | Arrow Left | Move the column selection to the left |
- | Ctrl + Enter | Used to select the multi cards |
- | Ctrl + Space | Used to select the multi cards |
- | Shift + Arrow Up | Used to select the multiple cards towards up |
- | Shift + Arrow Down | Used to select the multiple cards towards down |
- | Shift + Tab | Reverse order of the tab action |
- | Enter | Open the selected cards |
- | Tab | To navigate the Kanban column |
- | Delete | To delete the selected cards |
- | ESC | Escape from the modified details |
- | Space | Used to open the card edit dialog based on the column selection |

Ensuring accessibility

The Kanban component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Kanban component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Kanban component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Header double click in Angular Kanban component

You can bind the header double click event using [dataBound](#) event at initial rendering. You can get the column header text when double click the headers.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component } from '@angular/core';
import { CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { DialogUtility } from '@syncfusion/ej2-popups';
import { closest } from '@syncfusion/ej2-base';
import { kanbanData } from '../datasource';
```

```

@Component({
  imports: [

    KanbanModule

  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings' (dataBound)='OnDataBound()' '>
    <e-columns>
      <e-column headerText='To do' keyField='Open'></e-column>
      <e-column headerText='In Progress'
keyField='InProgress'></e-column>
      <e-column headerText='Testing' keyField='Testing'></e-
column>
      <e-column headerText='Done' keyField='Close'></e-column>
    </e-columns>
  </ejs-kanban>`
})
export class AppComponent {
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  OnDataBound(): void {
    let headerEle: HTMLElement = document.querySelector('.e-header-row')
    as HTMLElement;
    headerEle.addEventListener("dblclick", function (e: Event) {
      let target = closest((<HTMLElement>e.target), '.e-header-
cells');
      DialogUtility.alert({
        title: 'Header',
        content: "Double clicked on " +
(<HTMLElement>target.querySelector('.e-header-text')).innerText + " header",
        showCloseIcon: true,
        closeOnEscape: true,
        animationSettings: { effect: 'Zoom' }
      });
    });
  }
}

```

MAIN.TS

```

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

```

Dynamically change columns in Angular Kanban component

You can dynamically change the Kanban columns by using the [columns](#) property.

In the below sample, you can dynamically change the [allowToggle](#) property at the particular column when you click on the button. You can also change the initially created columns to the new Kanban columns by using the [columns](#) property when you click on the button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { Component, ViewChild } from '@angular/core';
import { KanbanComponent, CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<button ej-button class="e-btn" id="particularColumn"
(click)='toggle()'>Enable Allow Toggle</button>
    <button ej-button class="e-btn" id="column"
(click)='change()'>Change Columns</button>
    <ejs-kanban #kanbanObj keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
      <e-columns>
        <e-column headerText='To do' keyField='Open'></e-column>
        <e-column headerText='In Progress'
keyField='InProgress'></e-column>
        <e-column headerText='Testing' keyField='Testing'></e-
column>
        <e-column headerText='Done' keyField='Close'></e-column>
      </e-columns>
    </ejs-kanban>`
})
export class AppComponent {
  @ViewChild("kanbanObj") kanbanObj?: KanbanComponent;
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public toggle = (): void => {
    (this.kanbanObj as KanbanComponent).columns[1].allowToggle = true;
  }
  public change = (): void => {
    (this.kanbanObj as KanbanComponent).columns = [
      { headerText: 'To Do', keyField: 'Open' },
      { headerText: 'Done', keyField: 'Close' }
    ]
  }
}
```

MAIN.TS

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

Filter cards in Angular Kanban component

You can filter the collection of cards from the `dataSource` and display it on the Kanban board by using the [query](#) property.

In the below sample, you can filter the cards based on the `where` query and display the filtered data to the Kanban board.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { DropDownListModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { KanbanComponent, CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { Query } from '@syncfusion/ej2-data';
import { ChangeEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { kanbanData } from './datasource';
@Component({
  imports: [
    KanbanModule,
    DropDownListModule
  ],
  standalone: true,
  selector: 'app-root',
  template: ` <ejs-dropdownlist #dropdown [dataSource]='priorityData'
width='250px'
      (change)='change($event)' index=0 placeholder='Select a
priority'></ejs-dropdownlist>
      <ejs-kanban #kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
        <e-columns>
          <e-column headerText='To do' keyField='Open'></e-column>
          <e-column headerText='In Progress'
keyField='InProgress'></e-column>
          <e-column headerText='Testing' keyField='Testing'></e-
column>
          <e-column headerText='Done' keyField='Close'></e-column>
        </e-columns>
      </ejs-kanban>`
})
export class AppComponent {
  @ViewChild('kanban') kanbanObj?: KanbanComponent;
  public data: Object[] = kanbanData;
  public cardSettings: CardSettingsModel = {
    contentField: 'Summary',
    headerField: 'Id'
  };
  public priorityData: string[] = ['None', 'High', 'Normal', 'Low'];
```

```

    change(args: ChangeEventArgs): void {
        let filterQuery: Query = new Query();
        if (args.value !== 'None') {
            filterQuery = new Query().where('Priority', 'equal',
args.value);
        }
        (this.kanbanObj as KanbanComponent).query = filterQuery;
    }
}

```

MAIN.TS

```

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

```

Search cards in Angular Kanban component

You can search the cards in Kanban by using the `query` property.

In the following sample, the searching operation starts as soon as you start typing characters in the external text box. It will search the cards based on the `Id` and `Summary` using the `search` query with `contains` operator.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { KanbanModule } from '@syncfusion/ej2-angular-kanban'
import { TextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
import { KanbanComponent, CardSettingsModel } from '@syncfusion/ej2-angular-kanban';
import { TextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { Query } from '@syncfusion/ej2-data';
import { kanbanData } from './datasource';
@Component({
  imports: [

    KanbanModule,
    TextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<table>
    <tbody>
      <td style="width: 200px">
        <ejs-textbox id="search" #search placeholder="Enter
search text" [showClearButton]="true" (keyup)="searchClick($event)"></ejs-
textbox>
      </td>
      <td>
        <button ejs-button class="e-btn" id="reset"
(click)='reset()'>Reset</button>
      </td>
    </tbody>

```

```

        </table>
        <ejs-kanban #kanban keyField='Status' [dataSource]='data'
[cardSettings]='cardSettings'>
            <e-columns>
                <e-column headerText='To do' keyField='Open'></e-column>
                <e-column headerText='In Progress'
keyField='InProgress'></e-column>
                <e-column headerText='Testing' keyField='Testing'></e-
column>
                <e-column headerText='Done' keyField='Close'></e-column>
            </e-columns>
        </ejs-kanban>`
    })
    export class AppComponent {
        @ViewChild('search') textBoxObj?: TextBoxComponent;
        @ViewChild('kanban') kanbanObj?: KanbanComponent;
        public data: Object[] = kanbanData;
        public cardSettings: CardSettingsModel = {
            contentField: 'Summary',
            headerField: 'Id'
        };
        searchClick(e: KeyboardEvent): void {
            let searchValue: string = (<HTMLInputElement>e.target).value;
            let searchQuery: Query = new Query();
            if (searchValue !== '') {
                searchQuery = new Query().search(searchValue, ['Id', 'Summary'],
'contains', true);
            }
            (this.kanbanObj as KanbanComponent).query = searchQuery;
        }
        reset(): void {
            (this.textBoxObj as TextBoxComponent).value = '';
            (this.kanbanObj as KanbanComponent).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));

```

Ej1 api migration in Angular Kanban component

This article describes the API migration process of Kanban component from Essential JS 1 to Essential JS 2.

Columns

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Default | **Property** : *columns* </br></br> <ej-kanban><e-kanban-columns></br></e-kanban-columns></ej-kanban> | **Property** : *columns* </br></br> <ejs-kanban><e-columns></br></e-columns></ejs-kanban></br> |

| Header Text | **Property** : *headerText* </br> </br> <ej-kanban><e-kanban-columns> </br><e-kanban-column headerText="Backlog"></br></e-kanban-column> | **Property** : *headerText* </br> </br> <ejs-kanban><e-columns></br> <e-column headerText='To do'></br></e-column></br></e-columns></ejs-kanban> |

| Key Field | **Property** : *key* </br></br> <ej-kanban><e-kanban-columns></br><e-kanban-column key="Open"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban> | **Property** : *keyField* </br></br> <ejs-kanban><e-columns></br><e-column keyField='Open'></br></e-column></br></e-columns></ejs-kanban> |

| Initial Collapsed
Columns | **Property** : *isCollapsed* </br></br> <ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br>[isCollapsed]="true"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> | **Property** : *isExpanded* </br></br> <ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br>allowToggle='true'></br>[isExpanded]='true'></br></e-column></br></e-columns></br></ejs-kanban> |

| Cell Add card button | **Property** : *showAddButton* </br> </br> <ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br>[showAddButton]="true"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> | **Property** : *showAddButton* </br> </br> <ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br>[showAddButton]='true'></br></e-column></br></e-columns></br></ejs-kanban> |

| Column card count | **Property** : *enableTotalCount* </br> </br> <ej-kanban>[enableTotalCount]="true"></br></ej-kanban></br> | **Property** : *showItemCount* </br> </br> <ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br>[showItemCount]='true'></br></e-column></br></e-columns></br></ejs-kanban> |

| Template | **Property** : *headerTemplate* </br></br> <ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br>headerTemplate="#template"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> | **Property** : *template* </br></br> <ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br>template='#headerTemplate'></br></e-column></br></e-columns></br></ejs-kanban> |

| Allow Drop | **Property** : *allowDrop* </br> </br> <ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br>[allowDrop]="false"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban> | **Property** : *allowDrop* </br></br> <ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br>[allowDrop]="false"></br></e-column></br></e-columns></br></ejs-kanban> |

| Allow Drag | **Property:** *allowDrag* </br> </br> <ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br> [allowDrag]="false"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban> | **Property:** *allowDrag* </br></br><ejs-kanban></br><e-columns></br><e-column headerText='To do'></br>keyField='Open'></br> [allowDrag]="false"></br></e-column></br></e-columns></br></ejs-kanban> |

| Total Count text | **Property:** *totalCount* </br> </br> </br><ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br>[totalCount]="totalcount"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> **TS**</br>export class AppComponent { </br>constructor() {</br> this.totalcount = </br>{</br> text: "Backlog Count" };</br> } </br>} | **Not Available** |

| Width | **Property:** *width*</br></br><ej-kanban></br><e-kanban-columns></br><e-kanban-column key="Open"></br>headerText="Backlog"></br> width="200"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> | **Not Available** |

| Visible | **Property:** *visible*</br></br><ej-kanban></br><e-kanban-columns></br><e-kanban-column></br> headerText="Backlog"></br> key="Open"></br>[visible]="false"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br> | **Not Available** |

| Add/Delete Columns | **Method:** *columns(column, key, </br> [action])*</br></br>**Add :**</br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.columns</br>("Review", "Review", "add");</br></br>**Delete:** </br>kanbanObj.columns</br>("Review", "Review", "remove");</br> | **Method:** *addColumn(columnOptions,</br> index)*</br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.addColumn({ </br>headerText: "Review",</br> keyField: "Review"</br>}, 2);</br></br>**Method:** *deleteColumn(index)* </br></br>kanbanObj.deleteColumn(2);

| Show Columns | **Method:** *showColumns(headerText)* </br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>showColumns("Testing"); | **Method:** *showColumn(key)* </br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.showColumn</br>("Testing"); |

| Hide Columns | **Method:** *hideColumns(headerText)* </br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>hideColumns("Testing"); | **Method:** *hideColumns(key)* </br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.hideColumn</br>("Testing"); |

| Get Visible</br>Column Names | **Method:** *getVisibleColumnNames()*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>getVisibleColumnNames(); | **Not Applicable** |

| Get Column
By Header Text | **Method:** *getColumnByHeaderText*</br>(*headerText*)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>getColumnByHeaderText</br>("Testing"); | **Not Applicable** |

| Get Column Data | **Not Applicable** | **Method:** *getColumnData*()</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>getColumnData();</br> |

| Triggers after
cell is click | **Event:** *cellClick*</br></br><ej-kanban #kanbanObj (cellClick)="cellClick(\$event)"></br></ej-kanban></br>**TS**</br> cellClick(event) {} | **Not Applicable** |

Cards

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Card unique field | **Property :** </br>*fields.primaryKey*</br></br><ej-kanban fields.primaryKey="Id"></br></ej-kanban></br> | **Property :** </br>*cardSettings.headerField*</br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>**TS**</br> public cardSettings:</br> CardSettingsModel = {</br>headerField: 'Id'</br> }; |

| Content | **Property:** </br>*fields.content* </br></br><ej-kanban fields.content="Summary"></br></ej-kanban></br> | **Property :** </br>*cardSettings.contentField*</br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>**TS**</br> public cardSettings:</br> CardSettingsModel = {</br>contentField: 'Summary'</br> }; |

| Tag | **Property:** </br>*fields.tag* </br></br><ej-kanban fields.tag="Tags"></br></ej-kanban></br> | **Property :** </br>*cardSettings.tagsField*</br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>**TS**</br> public cardSettings:</br> CardSettingsModel = {</br>tagsField: 'Tags';</br> } |

| Left border color | **Property:** </br>*fields.color*</br></br><ej-kanban fields.color="Type"></br>[cardSettings.colorMapping]</br>="color"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br>color = {</br>"#cb2027": "Bug,Story",</br>"#67ab47": "Improvement",</br>"#fbae19": "Epic",</br>"#6a5da8": "Others"</br> }; </br>}</br> | **Property:** </br>*cardSettings.grabberField*</br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>**TS**</br> public cardSettings:</br> CardSettingsModel = {</br>grabberField: "color"</br> }; |

| Header | **Property:** </br>*fields.title*</br></br><ej-kanban fields.title="Assignee"></br></ej-kanban></br> | Card Unique mapping
 field is displayed
on card header. |

| Image | **Property:** </br>*fields.imageUrl*</br></br><ej-kanban fields.imageUrl="ImgUrl"></br></ej-kanban></br> | **Not Applicable** |

| CSS class | **Not Applicable** | **Property :** </br>*cardSettings.footerCssField*</br></br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>**TS**</br> public cardSettings:</br> CardSettingsModel = {</br>footerCssField: "classNames"</br> }; |

| Template | **Property:** </br>*cardSettings.template*</br></br><ej-kanban cardSettings.template="#template"></br></ej-kanban></br> | **Property:**

```
</br>cardSettings.template</br></br><ej-kanban</br>[cardSettings]='cardSettings'></br><ej-kanban></br>TS</br> public cardSettings:</br> CardSettingsModel = {</br>template:
"#cardTemplate"</br> }; |
```

```
| Toggle Card | Method: toggleCard</br>($div or id)</br></br><ej-kanban #kanbanObj></br></br>
kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj:
KanbanComponent;</br>kanbanObj.</br>toggleCard("2");</br> | Not Applicable |
```

```
| Get Card Details | Not Applicable | Method: </br>getCardDetails(target)</br></br><ej-kanban
#kanbanObj></br></br><ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj:
KanbanComponent;</br>kanbanObj.</br>getCardDetails(</br>obj.element</br>.querySelector(".e-card")); |
```

```
| Get Selected Cards | Not Applicable | Method: </br>getSelectedCards()</br></br><ej-kanban
#kanbanObj></br></br><ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj:
KanbanComponent;</br>kanbanObj.</br>getSelectedCards(); |
```

```
| Card Click | Event: cardClick</br></br><ej-kanban #kanbanObj></br>
(cardClick)="cardClick($event)"></br></br><ej-kanban></br>TS</br> cardClick(event) {} | Event:
cardClick </br></br><ej-kanban #kanbanObj></br> (cardClick)="cardClick($event)"></br></br><ej-kanban></br>TS</br> cardClick(event) {} |
```

```
| Card Double Click | Event: cardDoubleClick</br></br><ej-kanban #kanbanObj></br>
(cardDoubleClick)</br>="cardDoubleClick($event)"></br></br><ej-kanban></br>TS</br>
cardDoubleClick(event) {} | Event: cardDoubleClick </br></br><ej-kanban #kanbanObj></br>
(cardDoubleClick)=</br>"cardDoubleClick($event)"></br></br><ej-kanban></br>TS</br>
cardDoubleClick(event) {} |
```

```
| Triggers when start</br>the drag | Event: cardDragStart</br></br><ej-kanban #kanbanObj></br>
(cardDragStart)="cardDragStart($event)"></br></br><ej-kanban></br>TS</br> cardDragStart(event) {}
| Event: dragStart</br></br><ej-kanban #kanbanObj></br>
(dragStart)="dragStart($event)"></br></br><ej-kanban></br>TS</br> dragStart(event) {} |
```

```
| Triggers when card</br>is dragged | Event: cardDrag</br></br><ej-kanban #kanbanObj></br>
(cardDrag)="cardDrag($event)"></br></br><ej-kanban></br>TS</br> cardDrag(event) {} | Event: drag
</br></br><ej-kanban #kanbanObj></br> (drag)="drag($event)"></br></br><ej-kanban></br>TS</br>
drag(event) {} |
```

```
| Triggers when card</br>dragging stops | Event: cardDragStop</br></br><ej-kanban
#kanbanObj></br> (cardDragStop)=</br>"cardDragStop($event)"></br></br><ej-kanban></br>TS</br>
cardDragStop(event) {} | Event: dragStop</br></br><ej-kanban #kanbanObj></br>
(dragStop)="dragStop($event)"></br></br><ej-kanban></br>TS</br> dragStop(event) {} |
```

```
| Triggers after save</br>the data when dropped | Event: cardDrop</br></br><ej-kanban
#kanbanObj></br> (cardDrop)="cardDrop($event)"></br></br><ej-kanban></br>TS</br>
cardDrop(event) {} | Not Applicable |
```

```
| Triggers after</br>cell is click | Event: cellClick</br></br><ej-kanban #kanbanObj></br>
(cellClick)="cellClick($event)"></br></br><ej-kanban></br>TS</br> cellClick(event) {} | Not Applicable |
```

```
| Triggers each</br>card rendered | Event: queryCellInfo</br></br><ej-kanban #kanbanObj></br>
(queryCellInfo)=</br>"queryCellInfo($event)"></br></br><ej-kanban></br>TS</br>
```

queryCellInfo(event) {} | **Event:** *cardRendered*

(cardRendered)=>"cardRendered(\$event)">TS

cardRendered(event) {} |

DataSource

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Card unique field | **Property :** *fields.primaryKey*

fields.primaryKey="Id">| **Property :**

cardSettings.headerField>[cardSettings]='cardSettings'>TS

public cardSettings: CardSettingsModel = {headerField: 'Id' };

| DataSource | **Property:** *dataSource*

[dataSource]="kanbanData">TS

export class AppComponent {

public kanbanData: any; constructor() { this.kanbanData = []; }

Method:

dataSource(datasource)>ej-

kanban>TS

@ViewChild('kanbanObj') kanbanObj:

KanbanComponent; kanbanObj.dataSource(newDataSource); | **Property:** *dataSource*

[dataSource]="kanbanData">TS

public kanbanData: Object[] = extend([], kanbanData, null, true) as

Object[];

Method: *dataSource(datasource)*>ej-

kanbanObj>TS

@ViewChild('kanbanObj') kanbanObj:

KanbanComponent; kanbanObj.dataSource(newDataSource); |

| Triggers before data load | **Event:** *load*

(load)="load(\$event)">TS

load(event) {} | **Event:**

dataBinding>ej-

(dataBinding)="dataBinding(\$event)">TS

dataBinding(event) {} |

| Triggers after data bounded | **Event:** *dataBound*

(dataBound)="dataBound(\$event)">TS

dataBound(event) {} |

Event: *dataBound*>ej-

(dataBound)="dataBound(\$event)">TS

dataBound(event) {} |

Common

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Drag And Drop | **Property:** *allowDragAndDrop*

[allowDragAndDrop]="true">| **Property:**

allowDragAndDrop>ej-

kanban>[allowDragAndDrop]= true>ej-

kanban>|

| Key Field | **Property:** *keyField*

keyField="Status">ej-

| **Property :** *keyField*>ej-

keyField="Status">ej-

| Title | **Property:** *allowTitle*

[allowTitle]="true">ej-

Not Applicable |

| **CssClass** | **Property:** `cssClass`

```
<ej-kanban cssClass="gradient-green">
```

```
</ej-kanban> | Property: cssClass
```

```
<ej-kanban cssClass="custom-class">
```

```
</ej-kanban>
```

| **Print** | **Property:** `allowPrinting`

```
<ej-kanban [allowPrinting]="true">
```

```
</ej-kanban>
```

Method: `print()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.print();
```

| **Not Applicable** |

| **Touch** | **Property:** `enableTouch`

```
<ej-kanban [enableTouch]="true">
```

```
</ej-kanban>
```

| **Not Applicable** |

| **Locale** | **Property:** `locale`

```
<ej-kanban locale="de-DE">
```

```
</ej-kanban>
```

Property: `locale`

```
<ej-kanban locale="de-DE">
```

```
</ej-kanban>
```

| **Query** | **Property:** `query`

```
<ej-kanban [query]="query">
```

```
</ej-kanban>
```

TS

```
export class AppComponent {
```

```
  constructor() {
```

```
    this.query =
```

```
ej.Query().select("");
```

```
  };
```

```
</br>
```

| **Property:** `query`

```
<ej-kanban [query]="query">
```

```
</ej-kanban>
```

TS

```
public query =
```

```
new Query().select("");
```

```
</br>
```

```
|
```

| **Refresh** | **Method:** `refresh([templateRefresh])`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.refresh();
```

| **Method:** `refresh()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.refresh();
```

```
</br>
```

```
|
```

| **Refresh Template** | **Method:** `refreshTemplate()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.refreshTemplate();
```

| **Not Applicable** |

| **Destroy** | **Method:** `destroy()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.destroy();
```

| **Method:** `destroy()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.destroy();
```

```
</br>
```

```
|
```

| **Get Header Table** | **Method:** `getHeaderTable()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.getHeaderTable();
```

| **Not Applicable** |

| **Show Spinner** | **Not Applicable** | **Method:** `showSpinner()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.showSpinner();
```

```
</br>
```

```
|
```

| **Hide Spinner** | **Not Applicable** | **Method:** `hideSpinner()`

```
<ej-kanban #kanbanObj>
```

```
</ej-kanban>
```

TS

```
@ViewChild('kanbanObj') kanbanObj: KanbanComponent;
```

```
kanbanObj.hideSpinner();
```

```
</br>
```

```
|
```

| **Triggers before every action** | **Event:** `actionBegin`

```
<ej-kanban #kanbanObj>
```

```
(actionBegin) =
```

```
"actionBegin($event)"
```

```
</ej-kanban>
```

TS

```
actionBegin(event) {}
```

```
|
```

```

Event: actionBegin</br></br><ejs-kanban #kanbanObj</br>
(actionBegin)=</br>"actionBegin($event)"></br></ejs-kanban></br>TS</br> actionBegin(event) {}
|

```

```
| Triggers on successful completion of actions | Event: actionComplete
#kanbanObj (actionComplete)= "actionComplete($event)"
kanban>TS actionComplete(event) {} | Event: actionComplete
#kanbanObj (actionComplete)= "actionComplete($event)"
kanban>TS actionComplete(event) {} |
```

```
| Triggers on</br>action failure | Event: </br>actionFailure</br></br><ej-kanban #kanbanObj</br>
(actionFailure)=</br>"actionFailure($event)"></br></ej-kanban></br>TS</br> actionFailure(event)
} | Event: actionFailure</br></br><ejs-kanban #kanbanObj</br>
(actionFailure)=</br>"actionFailure($event)"></br></ejs-kanban></br>TS</br>
actionFailure(event) {} |
```

```
| Triggers after</br>Kanban rendered | Event: create</br></br><ej-kanban #kanbanObj</br>
(create)="create($event)"></br></ej-kanban></br>TS</br> create(event) {} | Event:
created</br></br><ejs-kanban #kanbanObj</br> (created)="created($event)"></br></ejs-
kanban></br>TS</br> created(event) {} |
```

[illegible]

```
| Triggers when destroy | Event: destroy</br></br><ej-kanban #kanbanObj</br>
(destroy)="destroy($event)"></br></ej-kanban></br>TS</br> destroy(event) {} | Event:
destroy</br></br><ejs-kanban #kanbanObj</br> (destroy)="destroy($event)"></br></ejs-
kanban></br>TS</br> destroy(event) {} |
```

Swimlane

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

— — — — —

| Default | **Property:** `swimlaneKey`
`kanban`
`fields.swimlaneKey="Assignee">` | **Property:**
`keyField`
`</ej-kanban>` | **Property:**
`[swimlaneSettings]='swimlaneSettings'>`
`</ej-`
`kanban>`
TS `public swimlaneSettings:` `SwimlaneSettingsModel =` `{ keyField:`
`'Assignee' };` |

```
| Header | Property: headers</br></br><ej-kanban</br>[headers]="headers"></br></ej-kanban>TS</br>export class AppComponent { </br>constructor() </br>this.headers = </br>{</br>text: "Andrew",</br>key: "Andrew Fuller"};</br> } </br>} | Property: textField</br></br><ejs-kanban</br>[swimlaneSettings]='swimlaneSettings'></br></ejs-kanban></br>TS</br>public swimlaneSettings:</br>SwimlaneSettingsModel =</br> { </br>textField: "AssigneeName" };</br> |
```

```
| Drag And Drop | Property: allowDragAndDrop</br></br><ej-kanban</br>[swimlaneSettings]="swimlaneSettings"></br></ej-kanban></br>TS</br>export class AppComponent { </br>constructor() {</br>  this.swimlaneSettings = </br>{ allowDragAndDrop: true
```

</br> } </br> } | **Property:** *allowDragAndDrop*</br></br><ej-kanban></br>[swimlaneSettings]='swimlaneSettings'></br></ej-kanban></br>**TS**</br> public swimlaneSettings:</br> SwimlaneSettingsModel =</br> { </br>allowDragAndDrop: true };</br> |

| Card Count | **Property:** *showCount*</br></br><ej-kanban></br>[swimlaneSettings]="swimlaneSettings"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br>constructor() {</br> this.swimlaneSettings = </br>{ showCount: true }</br> } </br> } | **Property:** *showItemCount*</br></br><ej-kanban></br>[swimlaneSettings]='swimlaneSettings'></br></ej-kanban></br>**TS**</br> public swimlaneSettings:</br> SwimlaneSettingsModel =</br> { </br>showItemCount: true };</br> |

| Empty Row | **Property:** *showEmptySwimlane*</br></br><ej-kanban></br>[swimlaneSettings]="swimlaneSettings"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br>constructor() {</br> this.swimlaneSettings = </br>{ showEmptySwimlane: true }</br> } </br> } | **Property:** *showEmptyRow*</br></br><ej-kanban></br>[swimlaneSettings]='swimlaneSettings'></br></ej-kanban></br>**TS**</br> public swimlaneSettings:</br> SwimlaneSettingsModel =</br> { </br>showEmptyRow: true };</br> |

| Sorting | **Not Available** | **Property:** *sortDirection*</br></br><ej-kanban></br>[swimlaneSettings]='swimlaneSettings'></br></ej-kanban></br>**TS**</br> public swimlaneSettings:</br> SwimlaneSettingsModel =</br> { </br>sortDirection: </br>"Descending" };</br> |

| Expand All | **Method:** *expandAll()*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanSwimlane</br>.expandAll(); | **Not Applicable** |

| Collapse All | **Method:** *collapseAll()*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanSwimlane</br>.collapseAll(); | **Not Applicable** |

| Toggle | **Method:** *toggle(\$div or key)*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanSwimlane</br>.toggle(\$(".e-slexpandcollapse")</br>.eq(1)); | **Not Applicable** |

| Get Swimlane Data | **Not Applicable** | **Method:** *getSwimlaneData(keyField)*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>getSwimlaneData("Janet");</br> |

| Triggers before swimlane</br>icon click event | **Event:** *swimlaneClick*</br></br><ej-kanban #kanbanObj></br> (swimlaneClick)="swimlaneClick(\$event)"></br></ej-kanban></br>**TS**</br> swimlaneClick(event) {} | **Not Applicable** |

Stacked Headers

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Multiple stacked headers | **Property:** *stackedHeaderColumns*</br></br><ej-kanban></br>[stackedHeaderRows]=</br>"stackedHeaderRow"></br></ej-


```
kanban</br>TS</br>export class AppComponent { </br>constructor() {</br> this.stackedHeaderRow =
</br>[{ stackedHeaderColumns: [{ </br> headerText: "Status",</br>column: "Backlog,</br>In Progress,
Testing, </br>Done"}] },</br> { stackedHeaderColumns: [{</br> headerText: "Unresolved",</br>column:
"Backlog,</br>In Progress"}]}}]; | Not Applicable |
```

| Single Stacked Header | **Property:** *stackedHeaderColumns*</br></br><ej-kanban</br>[stackedHeaderRows]=</br>"stackedHeaderRow"></br></ej-kanban></br>TS</br>export class AppComponent { </br>constructor() {</br> this.stackedHeaderRow = </br>[{ stackedHeaderColumns: [{ </br> headerText: "Status",</br>column: "Backlog,</br>In Progress, Testing, </br>Done"}] },</br> { stackedHeaderColumns: [{</br>headerText: "Unresolved",</br>column: "Backlog,</br>In Progress"}]}}]; | **Property:** </br>*stackedHeaders*</br><ejs-kanban></br><e-stackedHeaders></br><e-stackedHeader text='To Do'</br> keyFields='Open, InProgress'></br></e-stackedHeader></br></e-stackedHeaders></br></ejs-kanban></br> |

WIP Validation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Constraints Type | **Property:** </br>*constraints.type*</br></br><ej-kanban></br><e-kanban-columns></br><e-kanban-column</br>headerText="Backlog",</br>key: "Open",</br>[constraints]="constraint"></br></e-kanban-column></br></ej-kanban></br>TS</br>export class AppComponent { </br>constraint = { max: '2' };</br>}; | **Property:** </br>*constraintType*</br></br><ejs-kanban</br>constraintType="swimlane"></br></ejs-kanban></br> |

| Maximum card Count</br>at particular</br>column/swimlane | **Property:** </br>*constraints.max*</br></br><ej-kanban></br><e-kanban-columns></br><e-kanban-column</br>headerText="Backlog",</br>key: "Open",</br>[constraints]="constraint"></br></e-kanban-column></br></ej-kanban></br>TS</br>export class AppComponent { </br>constraint = {</br>type: "swimlane",</br>max: 5};</br>}; | **Property:** </br>*maxCount*</br></br><ejs-kanban></br><e-columns></br><e-column headerText='Backlog'</br> keyField='Open' maxCount='5'></br></e-column></br></e-columns></br></ejs-kanban></br> |

| Minimum card Count</br>at particular column | **Property:**</br>*constraints.min*</br></br><ej-kanban></br><e-kanban-columns></br><e-kanban-column</br>headerText="Backlog",</br>key: "Open",</br>[constraints]="constraint"></br></e-kanban-column></br></e-kanban-columns></br></ej-kanban></br>TS</br>export class AppComponent { </br>constraint = {</br>type: "swimlane",</br>min: 2};</br>}; | **Property:** *minCount*</br><ejs-kanban></br><e-columns></br><e-column headerText='Backlog'</br> keyField='Open' minCount='2'></br></e-column></br></e-columns></br></ejs-kanban></br> |

Keyboard

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| KeyBoard | **Property:** </br>*allowKeyboardNavigation*</br></br><ej-kanban</br>[allowKeyboardNavigation]="true"></br></ej-kanban></br> | **Property:**

`</br>allowKeyboard</br></br><ejs-kanban </br> [allowKeyboard]="true"></br></ejs-kanban></br> |`

| Settings | **Property:** `</br>keySettings</br></br><ej-kanban [keySettings]='keySettings'></br></ej-kanban></br>TS</br>export class AppComponent { </br>keySettings = {</br> focus: "e",</br> insertCard: "45"</br>}</br>}; | Not Applicable |`

Toggle Columns

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** `</br>allowToggleColumn</br></br><ej-kanban</br>[allowToggleColumn]="true"></br></ej-kanban></br> | Property: </br>allowToggle</br></br><ejs-kanban></br><e-columns></br><e-column [allowToggle]="true"></br></e-column></br></e-columns></br></ejs-kanban></br> |`

| Toggle | **Method:** `toggleColumn</br>(headerText or $div)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br>kanbanObj: KanbanComponent;</br>kanbanObj.</br>toggleColumn</br>("Backlog"); | Not Applicable |`

Dialog Editing

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Fields | **Property:** `editItems</br></br><ej-kanban</br>[editSettings.editItems]="editItem"></br></ej-kanban></br>TS</br>export class AppComponent { </br> editItem = [] </br> }; | Property: fields</br></br><ejs-kanban</br>[dialogSettings]='dialogSettings'></br></br></ej-kanban></br>TS</br> public dialogSettings:</br> DialogSettingsModel = {</br>fields: [] }</br> |`

| Dialog Model | **Not Available** | **Property:** `model</br></br><ejs-kanban</br>[dialogSettings]='dialogSettings'></br></br></ej-kanban></br>TS</br> public dialogSettings:</br> DialogSettingsModel = {</br>model: {</br> width: 250 } }</br>`

| Template | **Property:** `dialogTemplate</br></br><ej-kanban</br>editSettings.dialogTemplate="#template"></br></ej-kanban></br> | Property: template</br></br><ejs-kanban></br><ng-template </br>#dialogSettingsTemplate</br>let-data></br></ng-template></br></ejs-kanban></br> |`

| Enable Editing | **Property:** `allowEditing</br></br><ej-kanban</br>editSettings.allowEditing="true"></br></ej-kanban></br> | In default allowed for editing. |`

| Enable Adding | **Property:** `allowAdding</br></br><ej-kanban</br>editSettings.allowAdding="true"></br></ej-kanban></br> | Adding applicable using column</br> show add button or</br>public method. |`

| Edit Mode | **Property:** `editMode</br></br><ej-kanban</br>editSettings.editMode="dialogtemplate"></br></ej-kanban></br> | Not Applicable |`

| External Form template | **Property:** `</br>externalFormTemplate</br></br><ej-kanban</br>editSettings.</br>externalFormTemplate=</br>"#template"></br></ej-kanban></br>` | **Not Applicable** |

| External Form Position | **Property:** `</br>externalFormPosition</br></br><ej-kanban</br>editSettings.</br>externalFormPosition</br>="bottom"></br></ej-kanban></br>` | **Not Applicable** |

| Add Card | **Method:** `</br>KanbanEdit.addCard</br>([primaryKey], [card])</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.addCard("2",</br>{ Id: 2, Status: Open});` | **Method:** `</br></br>addCard(cardData)</br><ejs-kanban #kanbanObj></br></ejs-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.addCard({ Id: 2,</br>Status: Open});` |

| Update Card | **Method:** `updateCard(key, data)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.updateCard(2, { Id: 2,</br>Status: Open});` | **Method:** `updateCard(cardData)</br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.updateCard({ Id: 2,</br>Status: Open});` |

| Delete Card | **Method:** `</br>KanbanEdit.deleteCard(key)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.deleteCard(2);` | **Method:** `deleteCard()</br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.deleteCard(2);` |

| Cancel Edit | **Method:** `</br>KanbanEdit.cancelEdit()</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.cancelEdit();` | **Not Available** |

| End Edit | **Method:** `</br>KanbanEdit.endEdit()</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.endEdit();` | **Not Available** |

| Start Edit | **Method:** `</br>KanbanEdit.startEdit</br>($div or key)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.startEdit(2);` | **Method:** `</br>openDialog(action, data)</br></br><ejs-kanban #kanbanObj></br></ejs-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.openDialog("Add");` |

| Set Validation | **Method:** `KanbanEdit</br>.setValidationToField(name, rules)</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>KanbanEdit</br>.setValidationToField</br>("Summary", { required: true });` | **Not Available** |

| Close Dialog | **Not Applicable** | **Method:** `closeDialog()`
`<ej-kanban #kanbanObj>`
`kanban>`
TS
`@ViewChild('kanbanObj')`
`kanbanObj:`
`KanbanComponent;`
`kanbanObj.closeDialog();` |

| Triggers before dialog Open | **Not Applicable** | **Event:** `dialogOpen`
`<ej-kanban #kanbanObj>`
`(dialogOpen)="dialogOpen($event)"`
`</ej-kanban>`
TS
`dialogOpen(event) {}` |

| Triggers when dialog close | **Not Applicable** | **Event:** `dialogClose`
`<ej-kanban #kanbanObj>`
`(dialogClose)="dialogClose($event)"`
`</ej-kanban>`
TS
`dialogClose(event) {}` |

| Triggers after the card is edited | **Event:** `endEdit`
`<ej-kanban #kanbanObj>`
`(endEdit)="endEdit($event)"`
`</ej-kanban>`
TS
`endEdit(event) {}` | **Not Applicable** |

| Triggers after the card is deleted | **Event:** `endDelete`
`<ej-kanban #kanbanObj>`
`(endDelete)="endDelete($event)"`
`</ej-kanban>`
TS
`endDelete(event) {}` | **Not Applicable** |

| Triggers before task is edited | **Event:** `beginEdit`
`<ej-kanban #kanbanObj>`
`(beginEdit)="beginEdit($event)"`
`</ej-kanban>`
TS
`beginEdit(event) {}` | **Not Applicable** |

Dialog Editing Fields

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Fields | **Property:** `editItems`
`<ej-kanban>`
`[editSettings.editItems]="editItem"`
`</ej-kanban>`
TS
`export class`
`AppComponent {`
`editItem = []`
`};` | **Property:** `fields`
`<ej-kanban>`
`[dialogSettings]='dialogSettings'`
`</ej-kanban>`
TS
`public`
`dialogSettings:`
`DialogSettingsModel = {`
`fields: []`
`}` |

| Mapping key | **Property:** `field`
`<ej-kanban>`
`[editSettings.editItems]="editItem"`
`</ej-kanban>`
TS
`export class`
`AppComponent {`
`editItem = [{`
`field: "Id"`
`}]`
`};` | **Property:** `key`
`<ej-kanban>`
`[dialogSettings]='dialogSettings'`
`</ej-kanban>`
TS
`public`
`dialogSettings:`
`DialogSettingsModel = {`
`fields: [{`
`key: "Id"`
`}]`
`}` |

| Label | **Not Applicable** | **Property:** `text`
`<ej-kanban>`
`[dialogSettings]='dialogSettings'`
`</ej-kanban>`
TS
`public`
`dialogSettings:`
`DialogSettingsModel = {`
`fields: [{`
`text: "ID",`
`key: "Id"`
`}]`
`}` |

| Type | **Property:** `editType`
`<ej-kanban>`
`[editSettings.editItems]="editItem"`
`</ej-kanban>`
TS
`export class`
`AppComponent {`
`editItem = [{`
`editType: ej.Kanban.`
`EditingType.String`
`}]`
`};` | **Property:** `type`
`<ej-kanban>`
`[dialogSettings]='dialogSettings'`
`</ej-kanban>`
TS
`public`
`dialogSettings:`
`DialogSettingsModel = {`
`fields: [{`
`type:`
`"TextBox",`
`key: "Id"`
`}]`
`}` |

| Validation Rules | **Property:** *validationRules*</br></br><ej-kanban></br>[editSettings.editItems]="editItem"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br> editItem = [{ </br>validationRules: </br>required: true} </br> }]; | **Property:** *validationRules*</br></br><ejs-kanban></br>[dialogSettings]='dialogSettings'></br></br></ej-kanban></br>**TS**</br> public dialogSettings:</br> DialogSettingsModel = {</br>fields: [{</br>validationRules: {</br>required: true}}] </br> } |

| Params | **Property:** *editParams*</br></br><ej-kanban></br>[editSettings.editItems]="editItem"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br> editItem = [{ </br>editParams: {</br> decimalPlaces: 2}</br> }]; | **Not Applicable** |

| Default value | **Property:** *defaultValue*</br></br><ej-kanban></br>[editSettings.editItems]="editItem"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br> editItem = [{ </br>defaultValue: "Open"</br> }]; | **Not Applicable** |

Tooltip

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** </br>*tooltipSettings.enable*</br></br><ej-kanban></br>tooltipSettings.enable="true"></br></ej-kanban></br> | **Property:**</br>*enableTooltip*</br></br><ejs-kanban [enableTooltip]="true"></br></ej-kanban></br> |

| Template | **Property:** </br>*tooltipSettings.template*</br></br><ej-kanban></br>tooltipSettings.template="#tooltipTemplate"></br></ej-kanban></br> | **Property:** *tooltipTemplate*</br></br><ejs-kanban></br>tooltipTemplate="#tooltipTemplate"></br></ej-kanban></br> |

Context Menu

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** *enable*</br></br><ej-kanban></br>contextMenuSettings.enable="true"></br></ej-kanban></br> | **Not Applicable** |

| Menu Items | **Property:** *menuItems*</br></br><ej-kanban></br>contextMenuSettings.enable</br>="true"</br>[contextmenusettings.</br>menuItems]="menuItem"></br></ej-kanban></br>**TS**</br>this.menuItem = ["Move Right"];</br> | **Not Applicable** |

| Disable default Items | **Property:** *disableDefaultItems*</br></br><ej-kanban></br>contextMenuSettings.enable</br>="true"</br>contextmenusettings.</br>disableDefaultItems=</br>[ej.Kanban.MenuItem.MoveLeft]></br></ej-kanban></br> | **Not Applicable** |

| Custom Menu Items | **Property:** *customMenuItems*</br></br>**Text:**<ej-kanban></br>contextMenuSettings.enable</br>="true"</br>contextmenusettings.</br>custommenuItems=</br>"customMenuItems"</br></ej-

```
kanban</br>TS</br>export class AppComponent { </br> this.customMenuItems =</br> [{ text:
"Menu1" </br>}]</br></br>Target:</br><ej-
kanban</br>contextMenuSettings.enable</br>="true"</br>
contextmenuSettings.</br>custommenuItems=</br>"customMenuItems"</br></ej-
kanban</br>TS</br>export class AppComponent { </br> this.customMenuItems =</br> [{ target:
ej.Kanban</br>.Target.Header </br>}]</br></br>Template:</br><ej-
kanban</br>contextMenuSettings.enable</br>="true"</br>
contextmenuSettings.</br>custommenuItems=</br>"customMenuItems"</br></ej-
kanban</br>TS</br>export class AppComponent { </br> this.customMenuItems =</br> [{ text: "Hide
Column",</br>template: "#template"</br>}]</br>}; | Not Applicable |
```

| Triggers when context</br>menu item click | **Event:** *contextClick*</br><ej-kanban #kanbanObj</br>(contextClick)="contextClick(\$event)"></br></ej-kanban></br>TS</br> contextClick(event) {}> |
Not Applicable |

| Triggers when context</br>menu open | **Event:** *contextOpen*</br><ej-kanban #kanbanObj</br>(contextOpen)="contextOpen(\$event)"></br></ej-kanban></br>TS</br> contextOpen(event) {}> |
Not Applicable |

WorkFlows

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** *workFlows*</br></br><ej-kanban</br>[workflows]="workflow"></br></ej-kanban></br>TS</br>export class AppComponent { </br> workflow= [{}] </br> }; | **Not Applicable** |

| Key | **Property:** *key*</br></br><ej-kanban</br>[workflows]="workflow"></br></ej-kanban></br>TS</br>export class AppComponent { </br> workflow= [{</br>key: "Order"}]</br> }; |
Not Applicable |

| Allowed Transition | **Property:** *allowedTransition*</br></br><ej-kanban</br>[workflows]="workflow"></br></ej-kanban></br>TS</br>export class AppComponent { </br> workflow= [{</br>key: "Order", </br>allowedTransitions: "Served"</br>}]</br> }; | **Not Applicable** |

Filtering

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** *filterSettings*</br></br><ej-kanban</br>[filterSettings]="filterSetting"></br></ej-kanban></br>TS</br>export class AppComponent { </br> filterSetting= [] </br> }; | **Not Applicable** |

| Enable | **Property:** *allowFiltering*</br></br><ej-kanban</br>[allowFiltering]="true"></br></ej-kanban> | **Not Applicable** |

| Text | **Property:** *text*</br></br><ej-kanban</br>[filterSettings]="filterSetting"></br></ej-kanban></br>TS</br>export class AppComponent { </br> filterSetting= [{</br>text: "Janet Issues"</br>}] </br> }; | **Not Applicable** |

| Query | **Property:** *query*
`<ej-kanban [filterSettings]="filterSetting"></ej-kanban>`
TS
 export class AppComponent {
 filterSetting= [{
 query: new
 ej.Query().where("Assignee", "equal", "Janet")
 }]
 }; | **Not Applicable** |

| Description | **Property:** *description*
`<ej-kanban [filterSettings]="filterSetting"></ej-kanban>`
TS
 export class
 AppComponent {
 filterSetting= [{
 description: "Display Issues"
 }]
 }; | **Not Applicable** |

| Filter Cards | **Method:** *filterCards(query)*
`<ej-kanban #kanbanObj></ej-kanban>`
TS
 @ViewChild('kanbanObj') kanbanObj:
 KanbanComponent;
 kanbanObj.KanbanFilter.filterCards(new
 ej.Query().where("Assignee", "equal", "Janet")); | **Not Applicable** |

| Clear | **Method:** *clearFilter()*
`<ej-kanban #kanbanObj></ej-kanban>`
TS
 @ViewChild('kanbanObj') kanbanObj:
 KanbanComponent;
 kanbanObj.KanbanFilter.clearFilter(); | **Not Applicable** |

Searching

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** *searchSettings*
`<ej-kanban [searchSettings]="searchSettings"></ej-kanban>`
TS
 export class
 AppComponent {
 searchSettings= []
 }; | **Not Applicable** |

| Enable | **Property:** *allowSearching*
`<ej-kanban [allowSearching]="true"></ej-kanban>` | **Not Applicable** |

| Fields | **Property:** *fields*
`<ej-kanban [searchSettings]="searchSettings"></ej-kanban>`
TS
 export class AppComponent {
 searchSettings= [{
 fields: ["Summary"]
 }]
 }; | **Not Applicable** |

| Key | **Property:** *key*
`<ej-kanban [searchSettings]="searchSettings"></ej-kanban>`
TS
 export class AppComponent {
 searchSettings= [{
 key: "Task 1"
 }]
 }; | **Not Applicable** |

| Operator | **Property:** *operator*
`<ej-kanban [searchSettings]="searchSettings"></ej-kanban>`
TS
 export class
 AppComponent {
 searchSettings= [{
 operator: "contains"
 }]
 }; | **Not Applicable** |

| Ignore Case | **Property:** *ignoreCase*
`<ej-kanban [searchSettings]="searchSettings"></ej-kanban>`
TS
 export class
 AppComponent {
 searchSettings= [{
 ignoreCase: true
 }]
 }; | **Not Applicable** |

| Search Cards | **Method:** *searchCards(searchString)*
`<ej-kanban #kanbanObj></ej-kanban>`
TS
 @ViewChild('kanbanObj') kanbanObj:
 KanbanComponent;
 kanbanObj.KanbanFilter.searchCards("Analyze"); | **Not Applicable** |

| Clear | **Method:** *clearSearch()*
`<ej-kanban #kanbanObj></ej-kanban>`
TS
 @ViewChild('kanbanObj') kanbanObj:
 KanbanComponent;
 kanbanObj.KanbanFilter.clearSearch(); | **Not Applicable** |

External Drag And Drop

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** `</br>allowExternalDragAndDrop</br></br><ej-kanban</br>[allowExternalDragAndDrop]="true"></br></ej-kanban></br>` | **Not Applicable** |

| Target | **Property:** `</br>externalDropTarget</br></br><ej-kanban</br>[cardSettings]="cardSettings"></br></ej-kanban></br>TS</br>export class AppComponent { </br> cardSettings= {</br>externalDropTarget:</br>"#DroppedKanban" } </br> }; | Not Applicable |`

Scrolling

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** `allowScrolling</br></br><ej-kanban</br>[allowScrolling]="true"></br></ej-kanban></br>` | **Not Applicable** |

| height | **Property:** `height</br></br><ej-kanban</br>[allowScrolling]="true" [scrollSettings]="scrollSettings"></br></ej-kanban></br>TS</br>export class AppComponent { </br> scrollSettings= {</br>height: 400 } </br> }; | Property: height</br></br><ejs-kanban</br>height=400></br></br></ej-kanban></br> |`

| width | **Property:** `width</br></br><ej-kanban</br>[allowScrolling]="true" [scrollSettings]="scrollSettings"></br></ej-kanban></br>TS</br>export class AppComponent { </br> scrollSettings= {</br>width: 400 } </br> }; | Property: width</br></br><ejs-kanban</br>width=400></br></br></ej-kanban></br> |`

| Freeze Swimlane | **Property:** `allowFreezeSwimlane</br></br><ej-kanban</br>[allowScrolling]="true" [scrollSettings]="scrollSettings"></br></ej-kanban></br>TS</br>export class AppComponent { </br> scrollSettings= {</br>allowFreezeSwimlane: true } </br> }; | Not Applicable |`

| Get Scroll Object | **Method:** `</br>getScrollObject()</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>TS</br>@ViewChild('kanbanObj')</br> kanbanObj: KanbanComponent;</br>kanbanObj.</br>getScrollObject(); | Not Applicable |`

Card Selection and Hover

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Enable | **Property:** `allowSelection</br></br><ej-kanban</br>[allowSelection]="true"></br></ej-kanban></br>` | **Property:** `cardSettings.</br>selectionType</br></br><ejs-kanban</br>[cardSettings]='cardSettings'></br></ej-kanban></br>TS</br>public cardSettings:</br> CardSettingsModel = {</br>selectionType: "Single"</br> }; |`

| Type | **Property:** `selectionType</br></br><ej-kanban</br>selectionType="single"></br></ej-kanban></br>` | It is covered under `</br>selectionType` property. |

| Hover | **Property:** *allowHover*</br></br><ej-kanban</br>[allowHover]="true"></br></ej-kanban></br> | **Not Applicable** |

| Clear | **Method:** *clear()*</br></br><ej-kanban #kanbanObj></br></ej-kanban></br>**TS**</br>@ViewChild('kanbanObj')</br> kanbanObj:
KanbanComponent;</br>kanbanObj.KanbanSelection</br>.clear(); | **Not Applicable** |

| Triggers before</br>card selected | **Event:** *beforeCardSelect*</br></br><ej-kanban #kanbanObj</br>(beforeCardSelect)=</br>"beforeCardSelect(\$event)"></br></ej-kanban></br>**TS**</br>beforeCardSelect(event) {}></br></br>**Event:** *cardSelecting* </br><ej-kanban #kanbanObj</br>(cardSelecting)=</br>"cardSelecting(\$event)"></br></ej-kanban></br>**TS**</br>cardSelecting(event) {}> | **Not Applicable** |

| Triggers after</br>card selected | **Event:** *cardSelect*</br></br><ej-kanban #kanbanObj</br>(cardSelect)=</br>"cardSelect(\$event)"></br></ej-kanban></br>**TS**</br>cardSelect(event) {}> | **Not Applicable** |

Toolbar

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Custom Toolbar | **Property:** </br>*customToolbarItems.template*</br></br><ej-kanban</br>[customToolbarItems]="customToolbarItems"></br></ej-kanban></br>**TS**</br>export class AppComponent { </br> customToolbarItems= {</br>template:
"#Delete"</br>} </br> }; | **Not Applicable** |

| Triggers toolbar</br>item click | **Event:** *toolbarClick*</br></br><ej-kanban #kanbanObj</br>(toolbarClick)="toolbarClick(\$event)"></br></ej-kanban></br>**TS**</br>toolbarClick(event) {}> | **Not Applicable** |

Responsive

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Default | **Property:** *isResponsive*</br></br><ej-kanban</br>[isResponsive]="true"></br></ej-kanban> | **Not Applicable** |

| Minimum width | **Property:** *minWidth*</br></br><ej-kanban</br>[isResponsive]="true"</br>minWidth='400'></br></ej-kanban> | **Not Applicable** |

State Persistence

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Persistence | **Not Applicable** | **Property:**</br>*enablePersistence*</br></br><ejs-kanban
[enablePersistence]="true"></br></ejs-kanban></br> |

Right to Left - RTL

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

```
| default | Property: enableRTL</br></br><ej-kanban</br>[enableRTL]="true"></br></ej-kanban> |
Property: enableRtl</br></br><ejs-kanban [enableRtl]="true"></br></ejs-kanban></br> |
```

Linear Gauge

Getting started with Angular Linear gauge component

```
<!-- markdownlint-disable MD013 -->
```

This section explains the steps required to create a simple Linear Gauge and demonstrate the basic usage of the Linear Gauge component.

Dependencies

Below is the list of minimum dependencies required to use the Linear Gauge component.

```
`javascript
|-- @syncfusion/ej2-angular-lineargauge
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-angular-lineargauge
|-- @syncfusion/ej2-lineargauge
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-svg-base
`,`
```

Setup Angular Environment

[Angular CLI](#) can be used to setup the Angular applications. To install Angular CLI use the following command.

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

Create an Angular Application

Start a new Angular application using below Angular CLI command.

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

Installing Syncfusion Linear Gauge 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-lineargauge](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-lineargauge --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-lineargauge@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-lineargauge@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-lineargauge:"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 LinearGauge Module

Import **LinearGaugeModule** into Angular application in the `src/app/app.module.ts` file from the package `@syncfusion/ej2-angular-lineargauge`.

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the LinearGaugeModule for the LinearGauge component
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of Linear Gauge module in NgModule
  imports: [ BrowserModule, LinearGaugeModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
```

```

})
export class AppModule { }
,

```

- Modify the template in **app.component.ts** file to render the Linear Gauge component.

[src/app/app.component.ts].

```

`javascript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the LinearGauge component
  template: <ejs-lineargauge id='linear-container'></ejs-lineargauge>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
,

```

Now use the `app-container` in the index.html instead of default one.

```

<app-container></app-container>
,

```

- Now run the application in the browser using the below command.

```

npm start
,

```

The below example shows a basic Linear Gauge.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,

```

```

    selector: 'app-container',
    // specifies the template string for the linear gauge component
    template: `<ejs-lineargauge id="gauge-container"></ejs-lineargauge>`
  })
  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

LinearGauge component is segregated into the individual feature-wise modules. In order to use a particular feature, inject its feature module using the `providers: {}`. Please find the feature module name and description as follows.

- `AnnotationsService` - Inject this provider to use Annotation feature.
- `GaugeTooltipService` - Inject this provider to use Tooltip feature.

These modules should be injected in the providers section of the **app.module.ts** file as follows,

```

`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { AnnotationsService, GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge';
@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [AppComponent, LinearGaugeComponent],
  bootstrap: [AppComponent],
  providers: [ AnnotationsService, GaugeTooltipService ]
})
`

```

Add Gauge Title

The title can be added to the Linear Gauge component using the [title](#) property in the Linear Gauge.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-lineargauge id="gauge-container" [title]='Title'>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public Title?: string;
  ngOnInit(): void {
    // Title for linear gauge
    this.Title = 'linear gauge';
  }
}
```

MAIN.TS

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

Axis Range

The range of the axis can be set using the [minimum](#) and [maximum](#) properties in the Linear Gauge.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [minimum]='Minimum' [maximum]='Maximum'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
```

```
export class AppComponent implements OnInit {
  public Minimum?: number;
  public Maximum?: number;
  ngOnInit(): void {
    this.Minimum = 0,
    this.Maximum = 200
  }
}
```

MAIN.TS

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

To denote the axis labels with temperature units, add the °C as suffix to each label. This can be achieved by setting the **{value}°C** to the [format](#) property in the [labelStyle](#) object of the axis. Here, **{value}** acts as a placeholder for each axis label.

To change the pointer value from the default value of the gauge, set the [value](#) property in [pointers](#) object of the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis minimum=0 maximum=200>
          <e-pointers>
            <e-pointer value=140></e-pointer>
          </e-pointers>
          <e-ranges>
            <e-range start=0 end=80 startWidth=15 endWidth=15></e-range>
            <e-range start=80 end=120 startWidth=15 endWidth=15></e-range>
            <e-range start=120 end=140 startWidth=15 endWidth=15></e-range>
            <e-range start=140 end=200 startWidth=15 endWidth=15></e-range>
          </e-ranges>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public Label?: Object;
```

```

ngOnInit(): void {
  this.Label = {
    format: '{value}°C'
  };
}
}

```

MAIN.TS

```

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

```

Setting the value of pointer

The pointer value is changed in the below sample using the [value](#) property in [pointers](#) object of the axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-lineargauge id="gauge-container">
  <e-axes>
    <e-axis minimum=0 maximum=200>
      <e-pointers>
        <e-pointer value=40 color='green'></e-pointer>
      </e-pointers>
    </e-axis>
  </e-axes>
</ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}

```

MAIN.TS

```

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

```

Dimensions in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Size for Linear Gauge

The height and width of the Linear Gauge can be set using the [width](#) and [height](#) properties in [ejs-lineargauge](#).

In Pixel

<!-- markdownlint-disable MD036 -->

The size of the Linear Gauge can be set in pixel as demonstrated below.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-lineargauge id="gauge-container" width='350px'
height='100px'></ejs-lineargauge>`
})
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));
```

In Percentage

By setting value in percentage, Linear Gauge receives its dimension matching to its parent. For example, when the height is set as **50%**, Linear Gauge renders to half of the parent height. The Linear Gauge will be responsive when the width is set as **100%**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
```

```
providers: [ GaugeTooltipService ],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-lineargauge id="gauge-container" width='100%' height='50%'></ejs-
lineargauge>`
}))
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));
```

Note: When the component's size is not specified, the height will be **450px** and the width will be the same as the parent element's width.

Axis in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Axis is used to indicate the numeric values in the linear scale. The Linear Gauge component can have any number of axes. The sub-elements of an axis are line, ticks, labels, ranges, and pointers.

Setting the start value and end value of the axis

The start value and end value for the Linear Gauge can be set using the [minimum](#) and [maximum](#) properties in the [e-axis](#) respectively. By default, the start value of the axis is **0** and the end value of the axis is **100**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis minimum=20 maximum=200>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
}))
export class AppComponent {
  ngOnInit(): void {
```



```
}
}
```

MAIN.TS

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

Line Customization

The following properties in the [line](#) can be used to customize the axis line in the Linear Gauge.

- [height](#) - To set the length of the axis line.
- [width](#) - To set the thickness of the axis line.
- [color](#) - To set the color of the axis line.
- [offset](#) - To render the axis line with the specified distance from the Linear Gauge.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [line]='line'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public line?:Object;
  ngOnInit(): void {
    this.line = {
      height: 150,
      width: 2,
      color: '#4286f4',
      offset: 2
    };
  }
}
```

MAIN.TS

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

Ticks Customization

Ticks are used to specify the interval in the axis. Ticks are of two types, major ticks and minor ticks. The following properties in the [majorTicks](#) and [minorTicks](#) can be used to customize the major ticks and minor ticks respectively.

- [height](#) - To set the length of the major and minor ticks in pixel values.
- [color](#) - To set the color of the major and minor ticks of the Linear Gauge.
- [width](#) - To set the thickness of the major and minor ticks in pixel values.
- [interval](#) - To set the interval for the major ticks and minor ticks in the Linear Gauge.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis minimum=20 maximum=140 [majorTicks]='major'
[minorTicks]='minor'>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
})
export class AppComponent {
  public major?:Object;
  public minor?:Object;
  ngOnInit(): void {
    this.major = {
      interval: 20,
      color: "Orange"
    };
    this.minor = {
      interval: 5,
      color: 'red'
    };
  }
}
```

MAIN.TS

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

Positioning the ticks

The minor and major ticks can be positioned by using the [offset](#) and [position](#) properties. The [offset](#) is used to render the ticks with the specified distance from the axis. By default, the offset value is **0**. The possible values of the [position](#) property are **Inside**, **Outside**, **Cross**, and **Auto**. By default, the ticks will be placed inside the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis minimum=20 maximum=140 [majorTicks]='major'
[minorTicks]='minor'>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
})
export class AppComponent {
  public major?:Object;
  public minor?:Object;
  ngOnInit(): void {
    this.major = {
      interval: 20,
      color: "Orange",
      position: "Outside"
    };
    this.minor = {
      interval: 5,
      color: 'red',
      position: "Cross"
    };
  }
}
```

MAIN.TS

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

Labels Customization

The style of the labels can be customized using the following properties in the [font](#) property in [labelStyle](#).

- [color](#) - To set the color of the axis label.
- [fontFamily](#) - To set the font family of the axis label.
- [fontStyle](#) - To set the font style of the axis label.
- [fontWeight](#) - To set the font weight of the axis label.
- [opacity](#) - To set the opacity of the axis label.
- [size](#) - To set the size of the axis label.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [labelStyle]='label'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public label?:Object;
  ngOnInit(): void {
    this.label = {
      font: {
        color: 'red'
      }
    };
  }
}
```

MAIN.TS

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

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

Positioning the axis label

Labels can be positioned by using [offset](#) and [position](#) properties in the [labelStyle](#). The [offset](#) defines the distance between the labels and ticks. By default, the offset value is **0**. The possible values of the [position](#) property are **Inside**, **Outside**, **Cross**, and **Auto**. By default, the labels will be placed inside the axis.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [labelStyle]='label'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public label?:Object;
  ngOnInit(): void {
    this.label = {
      position: "Cross"
    };
  }
}
```

MAIN.TS

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

Customizing the display of the last label

If the last label is not in the visible range, it will be hidden by default. The last label can be made visible by setting the [showLastLabel](#) property as **true** in the [e-axis](#).

APP.COMPONENT.TS

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

```
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [showLastLabel]='isLabel' [maximum]='maximum'>
          <e-pointers>
            <e-pointer value=20></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public maximum?: number;
  public isLabel?: boolean;
  ngOnInit(): void {
    this.maximum = 115;
    this.isLabel = true;
  }
}
```

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 -->

Label Format

Axis labels in the Linear Gauge control can be formatted using the [format](#) property in the [labelStyle](#). It is used to render the axis labels in a certain format or to add a user-defined unit in the label. It works with the help of placeholder like **{value}°C**, where **value** represents the axis value. For example, 20°C.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
```

```

standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [labelStyle]='label'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
export class AppComponent {
  public label?:Object;
  ngOnInit(): void {
    this.label = {
      format: '{value}°C'
    };
  }
}

```

MAIN.TS

```

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

```

Displaying numeric format in labels

The numeric formats such as currency, percentage, and so on can be displayed in the labels of the Linear Gauge using the [format](#) property in the [ejs-lineargauge](#). The following table describes the result of applying some commonly used label formats on numeric values.

<!-- markdownlint-disable MD033 -->

Label Value	Label Format property value	Result	Description
1000	n1	1000.0	The number is rounded to 1 decimal place.
1000	n2	1000.00	The number is rounded to 2 decimal place.
1000	n3	1000.000	The number is rounded to 3 decimal place.
0.01	p1	1.0%	The number is converted to percentage with 1 decimal place.
0.01	p2	1.00%	The number is converted to percentage with 2 decimal place.
0.01	p3	1.000%	The number is converted to percentage with 3 decimal place.
1000	c1	\$1,000.0	The currency symbol is appended to number and number is rounded to 1 decimal place.

1000	c2	\$1,000.00	The currency symbol is appended to number and number is rounded to 2 decimal place.
------	----	------------	-------------------------------------------------------------------------------------

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" format="c">
      <e-axes>
        <e-axis minimum=20 maximum=140>
      </e-axis>
    </e-axes>
  </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Orientation

By default, the Linear Gauge is rendered vertically. To change its orientation, the [orientation](#) property must be set to "Horizontal".

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
```



```

    selector: 'app-container',
    template: `
    <ejs-lineargauge id="gauge-container" orientation='Horizontal'>
      <e-axes>
        <e-axis minimum=20 maximum=140>
      </e-axis>
    </e-axes>
    </ejs-lineargauge>`
  })
  export class AppComponent {
    ngOnInit(): void {
    }
  }
}

```

MAIN.TS

```

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

```

Inverted Axis

The axis of the Linear Gauge component can be inverted by setting the [isInversed](#) property to **true** in the [e-axis](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis isInversed=true>
      </e-axis>
    </e-axes>
    </ejs-lineargauge>`
  })
  export class AppComponent {
    ngOnInit(): void {
    }
  }
}

```

MAIN.TS

```

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

```

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

Opposed Axis

To place an axis opposite from its original position, [opposedPosition](#) property in the [e-axis](#) must be set as **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis opposedPosition=true>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Multiple Axes

Multiple axes can be added to the Linear Gauge by adding multiple [e-axis](#) in the [e-axes](#) and customization can be done with the [e-axis](#). Each axis can be customized separately as shown in the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
```

```

@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [labelStyle]='label1'>
        </e-axis>
        <e-axis opposedPosition=true [labelStyle]='label2'>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public label1?:Object;
  public label2?:Object;
  ngOnInit(): void {
    this.label1 = {
      format: '{value}°C'
    };
    this.label2 = {
      format: '{value}°F'
    };
  }
}

```

MAIN.TS

```

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

```

Ranges in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Range is the set of values in the axis. The range can be defined using the [start](#) and [end](#) properties in the [e-range](#). Any number of ranges can be added to the Linear Gauge using the [e-ranges](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],

```

```

standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-ranges>
            <e-range start=50 end=80></e-range>
          </e-ranges>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
export class AppComponent {
  ngOnInit(): void {
  }
}

```

MAIN.TS

```

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

```

Customizing the range

Ranges can be customized using the following properties in [e-range](#).

- [startWidth](#) - Customize the range thickness at the start axis value.
- [endWidth](#) - Customize the range thickness at the end axis value.
- [color](#) - Customize the range color.
- [position](#) - To place the range. By default, the range is placed outside of the axis. To change the position, this property can be set as "Inside", "Outside", "Cross", or "Auto".
- [Offset](#) - To place the range with specified distance from the axis.
- [border](#) - Customize color and width of range border.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">

```

```

        <e-axes>
          <e-axis>
            <e-ranges>
              <e-range start=50 end=80 startWidth=15 endWidth=15
color="Orange"></e-range>
            </e-ranges>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
    })
    export class AppComponent {
      ngOnInit(): void {
      }
    }
  }

```

MAIN.TS

```

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

```

Setting the range color for the labels

To set the color of the labels like the range color, set the [useRangeColor](#) property as **true** in the [labelStyle](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-
angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis [labelStyle]='labelStyle'>
          <e-ranges>
            <e-range start=50 end=80 startWidth=15 endWidth=15
color="red"></e-range>
          </e-ranges>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
  export class AppComponent {
    public labelStyle?: Object;
  }

```

```
ngOnInit(): void {
    this.labelStyle = {
        useRangeColor: true
    };
}
}
```

MAIN.TS

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

Multiple ranges

Multiple ranges can be added to the Linear Gauge by adding collections of [e-range](#) in the [e-ranges](#) and customization of ranges can be done with [e-range](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-
angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-ranges>
            <e-range start=0 end=30 startWidth=10 endWidth=10
color='#41f47f'></e-range>
            <e-range start=30 end=50 startWidth=10 endWidth=10
color='#f49441'></e-range>
            <e-range start=50 end=100 startWidth=10 endWidth=10
color='#cd41f4'></e-range>
          </e-ranges>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Gradient Color

Gradient support allows the addition of multiple colors in the range. The following gradient types are supported in the Linear Gauge.

- Linear Gradient
- Radial Gradient

Linear Gradient

Using linear-gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [startValue](#) property. The end value of the linear gradient will be set using the [endValue](#) property. The color stop values such as [color](#), [opacity](#), and [offset](#) to be defined in [colorStop](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-
angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'
orientation='horizontal' [axes]='axes'>
    </ejs-lineargauge>`
})
export class AppComponent {
  public container?: Object;
  public axes?: Object[];
  ngOnInit(): void {
    // Initialize objects
    this.container = { width: 30, offset: 30 }
    this.axes = [{
      minimum: 0,
      maximum: 100,
      line: {
        width: 0
      },
    },
    majorTicks: {
      interval: 25,
      height: 0
    }
  ],
}
```

```

    minorTicks: {
      height: 0
    },
    labelStyle: {
      font: {
        color: '#424242',
      }, offset: 55
    },
    pointers: [{
      value: 80, height: 25,
      width: 35, placement: 'Near',
      offset: -44, markerType: 'Triangle',
      color: '#f54ea2'
    }],
    ranges: [{
      start: 0, end: 80,
      startWidth: 30, endWidth: 30,
      offset: 30,
      linearGradient: {
        startValue: '0%',
        endValue: '100%',
        colorStop: [
          { color: '#fef3f9', offset: '0%', opacity: 1 },
          { color: '#f54ea2', offset: '100%', opacity: 1 }
        ]
      }
    }
  ]];
}
}

```

MAIN.TS

```

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

```

Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [innerPosition](#) property. The outer circle position of the radial gradient can be set using the [outerPosition](#) property. The color stop values such as [color](#), [opacity](#), and [offset](#) to be defined in [colorStop](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],

```



```

providers: [ GaugeTooltipService, GradientService ],
standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'
orientation='Horizontal' [axes]='axes'>
    </ejs-lineargauge>`
  ))
export class AppComponent {
  public container?: Object;
  public axes?: Object[];
  ngOnInit(): void {
    // Initialize objects
    this.container = { width: 30, offset: 30 }
    this.axes = [{
      minimum: 0,
      maximum: 100,
      line: {
        width: 0
      },
      majorTicks: {
        interval: 25,
        height: 0
      },
      minorTicks: {
        height: 0
      },
      labelStyle: {
        font: {
          color: '#424242',
        }, offset: 55
      },
      pointers: [{
        value: 80, height: 25,
        width: 35, placement: 'Near',
        offset: -44, markerType: 'Triangle',
        color: '#f54ea2'
      }],
      ranges: [{
        start: 0, end: 80,
        startWidth: 30, endWidth: 30,
        offset: 30,
        radialGradient: {
          radius: '65%',
          outerPosition: { x: '50%', y: '70%' },
          innerPosition: { x: '60%', y: '60%' },
          colorStop: [
            { color: '#fff5f5', offset: '5%', opacity: 0.9 },
            { color: '#f54ea2', offset: '100%', opacity: 0.9 }
          ]
        }
      }
    ]
  }];
}

```

MAIN.TS

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

If we set both gradients for the range, only the linear gradient gets rendered. If we set the [startValue](#) and [endValue](#) property of the [linearGradient](#) as empty strings, then the radial gradient gets rendered in the range of the Linear Gauge.

Pointers in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Pointers are used to indicate values on an axis. The value of the pointer can be modified using the [value](#) property in [e-pointer](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=40></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Types of pointer

The Linear Gauge supports the following types of pointers.

- Bar
- Marker

The type of pointer can be modified by using the [type](#) property in [e-pointer](#).

Marker pointer

A marker pointer is a shape that can be used to mark the pointer value in the Linear Gauge.

Types of marker shapes

By default, the marker shape for the pointer is **InvertedTriangle**. To change the shape of the pointer, use the [markerType](#) property in [e-pointer](#). The following marker types are available in Linear Gauge.

- Circle
- Rectangle
- Triangle
- InvertedTriangle
- Diamond
- Image
- Text

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=80 markerType='Circle'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Image can be rendered instead of rendering a shape as a pointer. It can be achieved by setting the [markerType](#) property to **Image** and setting the source URL of image to [imageUrl](#) property in [e-pointer](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" orientation="Horizontal">
      <e-axes>
        <e-axis [majorTicks]='major' [minorTicks]='minor'
[labelStyle]='label'>
          <e-pointers>
            <e-pointer value=60 markerType="Image" width="40" height="40"
imageUrl="https://ej2.syncfusion.com/angular/demos/assets/linear-
gauge/images/step-count.png" offset="-27"></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`,
})
export class AppComponent {
  public major?: Object;
  public minor?: Object;
  public label?: Object;
  ngOnInit(): void {
    this.label = {
      position: 'Outside',
    };
    this.major = {
      interval: 20,
      position: 'Outside',
    };
    this.minor = {
      position: 'Outside',
    };
  }
}
```

MAIN.TS

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

Text can be added instead of rendering a shape as a pointer. It can be achieved by setting the [markerType](#) property to **Text**, and the text content can be set using the [text](#) property in [e-pointer](#).

The following properties in the [textStyle](#) property can be used to set the text style for the text pointer.

- [fontFamily](#) - It is used to set the font family for the text pointer.
- [fontStyle](#) - It is used to set the font style for the text pointer.
- [fontWeight](#) - It is used to set the font weight for the text pointer.
- [size](#) - It is used to set the font size for the text pointer.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" orientation="Horizontal">
      <e-axes>
        <e-axis [majorTicks]='majorTicks' [minorTicks]='minorTicks'
[labelStyle]='labelStyle' [line]='lineStyle">
          <e-pointers>
            <e-pointer value="13" markerType="Text" text="Low"
color="black" offset="-35" [textStyle]="textStyle"></e-pointer>
            <e-pointer value="48" markerType="Text" text="Moderate"
color="black" offset="-35" [textStyle]="textStyle"></e-pointer>
            <e-pointer value="83" markerType="Text" text="High"
color="black" offset="-35" [textStyle]="textStyle"></e-pointer>
          </e-pointers>
          <e-ranges>
            <e-range start=0 end=30 color='#6FC78A' startWidth=50
endWidth=50 position='Outside'></e-range>
            <e-range start=30 end=65 color='#ECC85B' startWidth=50
endWidth=50 position='Outside'></e-range>
            <e-range start=65 end=100 color='#FB7D55' startWidth=50
endWidth=50 position='Outside'></e-range>
          </e-ranges>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`,
})
```

```

export class AppComponent {
  public majorTicks?: Object;
  public minorTicks?: Object;
  public labelStyle?: Object;
  public textStyle?: Object;
  public lineStyle?: Object;
  ngOnInit(): void {
    this.lineStyle = {
      width: 0,
    };
    this.majorTicks = {
      interval: 20,
      height: 7,
      width: 1,
      position: 'Outside',
    };
    this.minorTicks = {
      interval: 10,
      height: 3,
      position: 'Outside',
    };
    this.textStyle = {
      size: '18px',
      fontWeight: 'bold',
    };
    this.labelStyle = {
      position: 'Outside',
      font: { fontFamily: 'inherit' },
    };
  }
}

```

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 -->

Marker Pointer Customization

The marker pointer can be customized using the following properties.

- [height](#) - To set the height of the pointer.
- [position](#) - The position of the pointer can be changed by setting the value as **Inside**, **Outside**, **Cross**, or **Auto**.
- [width](#) - To set the width of the pointer.
- [color](#) - To set the color of the pointer.
- [placement](#) - To place the pointer in the specified position. By default, the pointer is placed **Far** from the axis. To change the placement, set the [placement](#) property as **Near**, **Center**, or **None**.
- [offset](#) - To place the pointer with specified distance from the axis.
- [opacity](#) - To set the opacity of the pointer.

- [animationDuration](#) - To specify the duration of the animation in pointer.
- [border](#) - To set the color and width for the border of the pointer.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=80 markerType='Circle' [height]='height'
[width]='width' color='#cd41f4'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  public height?: number;
  public width?: number;
  ngOnInit(): void {
    this.height = 15;
    this.width= 15;
  }
}
```

MAIN.TS

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

Bar Pointer

The bar pointer is used to track the axis value. The bar pointer starts from the beginning of the gauge and ends at the pointer value. To enable bar pointer set the [type](#) property in [e-pointer](#) as **Bar**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
```

```
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=80 type='Bar'></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

<!-- markdownlint-disable MD013 -->

Bar pointer customization

The bar pointer can be customized using following properties.

- [width](#) - To set the thickness of the bar pointer.
- [color](#) - To set the color of the bar pointer.
- [offset](#) - To place the bar pointer with the specified distance from it's default position.
- [opacity](#) - To set the opacity of the bar pointer.
- [roundedCornerRadius](#) - To set the corner radius for the bar pointer.
- [border](#) - To set the color and width for the border of the pointer.
- [animationDuration](#) - To set the duration of the animation in bar pointer.

The placement property is not applicable for the bar pointer.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
```



```
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=60 type='Bar' width=20 color='#f44141'></e-
pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Multiple pointers

Multiple pointers can be added to the Linear Gauge by adding multiple [e-pointer](#) in the [e-pointers](#) and customization for the pointers can be done with [e-pointer](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
```

```

        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=80></e-pointer>
              <e-pointer value=60 markerType='Circle'></e-pointer>
              <e-pointer value=30 markerType='Diamond'></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
    })
    export class AppComponent {
      ngOnInit(): void {
      }
    }
  }
}

```

MAIN.TS

```

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

```

Pointer animation

Pointer is animated on loading the gauge. This can be handled by using the [animationDuration](#) property. The duration of the animation can be specified in milliseconds.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-
angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=60 animationDuration=1000></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
  export class AppComponent {
    ngOnInit(): void {

```

```
}
}
```

MAIN.TS

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

Gradient Color

Gradient support allows the addition of multiple colors in the pointers of the Linear Gauge. The following gradient types are supported in the Linear Gauge.

- Linear Gradient
- Radial Gradient

Linear Gradient

Using linear gradient, colors will be applied in a linear progression. The start value of the linear gradient can be set using the [startValue](#) property. The end value of the linear gradient will be set using the [endValue](#) property. The color stop values such as [color](#), [opacity](#) and [offset](#) are set using [colorStop](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'
    orientation='horizontal'>
      <e-axes>
        <e-axis minimum=0 maximum=100 [line]='line'
        [majorTicks]='majorTicks' [minorTicks]='minorTicks'
        [labelStyle]='labelStyle' [pointers]='pointers' [ranges]='ranges'>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
  })
export class AppComponent {
  public container?: Object;
  public line?: Object;
  public majorTicks?: Object;
  public minorTicks?: Object;
```

```

public labelStyle?: Object;
public pointers?: Object[];
public ranges?: Object[];
ngOnInit(): void {
  this.container = { width: 30, offset: 30 }
  this.line= {
    width: 0
  };
  this.majorTicks= {
    interval: 25,
    height: 0
  };
  this.minorTicks= {
    height: 0
  },
  this.labelStyle= {
    font: {
      color: '#424242',
      size: '0px'
    }, offset: 55
  };
  this.pointers = [{
    value: 80, height: 25,
    width: 35, placement: 'Near',
    offset: -44, markerType: 'Triangle',
    linearGradient: {
      startValue: '0%',
      endValue: '100%',
      colorStop: [
        { color: '#fef3f9', offset: '0%', opacity: 1 },
        { color: '#f54ea2', offset: '100%', opacity: 1 }
      ]
    }
  }
  ]];
  this.ranges=[{
    start: 0, end: 80,
    startWidth: 30, endWidth: 30,
    color: '#f54ea2', offset: 30
  }]
}
}

```

MAIN.TS

```

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

```

Radial Gradient

Using radial gradient, colors will be applied in circular progression. The inner circle position of the radial gradient will be set using the [innerPosition](#) property. The outer circle position of the radial gradient can be set using the [outerPosition](#) property. The color stop values such as [color](#), [opacity](#), and [offset](#) are set using [colorStop](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService, GradientService } from '@syncfusion/ej2-
angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService, GradientService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'
orientation='horizontal'>
      <e-axes>
        <e-axis minimum=0 maximum=100 [line]='line'
[majorTicks]='majorTicks' [minorTicks]='minorTicks'
[labelStyle]='labelStyle' [pointers]='pointers' [ranges]='ranges'>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
})
export class AppComponent {
  public container?: Object;
  public line?: Object;
  public majorTicks?: Object;
  public minorTicks?: Object;
  public labelStyle?: Object;
  public pointers?: Object[];
  public ranges?: Object[];
  ngOnInit(): void {
    this.container = { width: 30, offset: 30 }
    this.line= {
      width: 0
    };
    this.majorTicks= {
      interval: 25,
      height: 0
    };
    this.minorTicks= {
      height: 0
    },
    this.labelStyle= {
      font: {
        color: '#424242',
        size: '0px'
      }, offset: 55
    };
    this.pointers = [{
      value: 80, height: 25,
      width: 35, placement: 'Near',
      offset: -44, markerType: 'Triangle',
      radialGradient: {

```

```

        radius: '60%',
        outerPosition: { x: '50%', y: '50%' },
        innerPosition: { x: '50%', y: '50%' },
        colorStop: [
            { color: '#fff5f5', offset: '0%', opacity: 0.9 },
            { color: '#f54ea2', offset: '100%', opacity: 0.8 }
        ]
    }
}];
this.ranges=[{
    start: 0, end: 80,
    startWidth: 30, endWidth: 30,
    color: '#f54ea2', offset: 30
}]
}
}

```

MAIN.TS

```

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

```

If we set both gradients, only the linear gradient gets rendered. If we set the [startValue](#) and [endValue](#) property of the [linearGradient](#) as empty strings, then the radial gradient gets rendered in the pointer of the Linear Gauge.

Annotations in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Annotations are used to mark the specific area of interest in the Linear Gauge with text, HTML elements, or images. Any number of annotations can be added to the Linear Gauge component.

Adding annotation

To render the custom HTML elements in the Linear Gauge component, use the [content](#) property in the [e-annotation](#). The annotation can be rendered either by specifying the id of the element or specifying the code to create a new element that needs to be displayed in the gauge area.

<!-- markdownlint-disable MD036 -->

,

```
<script id='fruits' type='text/x-template'>
```

```
<div id='apple'>
```

```
<img src='src/lineargauge/images/apple.png'>
```

```
</div>
```

```
</script>
```

,

```
`typescript
```

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
    <e-annotations>
    <e-annotation zIndex='1' content='#fruits' x=100 y=-70></e-annotation>
    </e-annotations>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
,
```

Customization

The following properties are used to customize the annotation.

- [zIndex](#) - Bring the annotation to the front or back, when annotation overlaps with another element.
- [axisValue](#) - To place the annotation in the specified axis value with respect to the provided axis index.
- [axisIndex](#) - To place the annotation in the specified axis with respect to the provided axis value.
- [horizontalAlignment](#) - To place the annotation horizontally.
- [verticalAlignment](#) - To place the annotation vertically.
- [x](#), [y](#) - To place the annotation in the specified location.

Changing the z-index

To change the stack order of an annotation element, the [zIndex](#) property of the [e-annotation](#) can be used.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { AnnotationsService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ AnnotationsService ],
  standalone: true,
```

```

    selector: 'app-container',
    template: `
      <ejs-lineargauge id="gauge-container">
        <e-annotations>
          <e-annotation zIndex='1' content='<div
id="first"><h1>Gauge</h1></div>' x=100 y=-70 ></e-annotation>
        </e-annotations>
      </ejs-lineargauge>`
  })
  export class AppComponent {
    ngOnInit(): void {
    }
  }
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD036 -->

Positioning an annotation

The annotation can be placed anywhere in the Linear Gauge by setting the pixel value to the [x](#) and [y](#) properties in the [e-annotation](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { AnnotationsService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ AnnotationsService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-annotations>
        <e-annotation content='<div id="first"><h1>Gauge</h1></div>'
zIndex="1" x=250 y=-70></e-annotation>
      </e-annotations>
    </ejs-lineargauge>`
  })
  export class AppComponent {
    ngOnInit(): void {
    }
  }
}

```

MAIN.TS


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

<!-- markdownlint-disable MD036 -->

Alignment of annotation

The annotation can be aligned horizontally and vertically by using the [horizontalAlignment](#) and [verticalAlignment](#) properties respectively. The possible values can be **Center**, **Far**, **Near**, and **None**. The [horizontalAlignment](#) and [verticalAlignment](#) properties are not applicable when the [x](#) and [y](#) properties are set in the [e-annotation](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { AnnotationsService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ AnnotationsService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-annotations>
        <e-annotation content='<div id="first"><h1>Gauge</h1></div>'
zIndex="1" horizontalAlignment="Center" verticalAlignment="Center" ></e-
annotation>
      </e-annotations>
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Multiple annotations

Multiple annotations can be added to the Linear Gauge component by adding the multiple [e-annotation](#) in the [e-annotations](#) and customization for the annotation can be done with the [e-annotation](#).

APP.COMPONENT.TS

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

```
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { AnnotationsService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ AnnotationsService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container">
      <e-annotations>
        <e-annotation content='<div><h1
style="color:red;">Speed</h1></div>' zIndex="1" x=250 y=80>
        </e-annotation>
        <e-annotation content='<div><h1
style="color:blue;">Meter</h1></div>' zIndex="1" x=240 y=-100>
        </e-annotation>
      </e-annotations>
    </ejs-lineargauge>`
  })
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Animation in Angular Linear gauge component

All of the elements in the Linear Gauge, such as the axis lines, ticks, labels, ranges, pointers, and annotations, can be animated sequentially by using the [animationDuration](#) property. The animation for the Linear Gauge is enabled when the [animationDuration](#) property is set to an appropriate value in milliseconds, providing a smooth rendering effect for the component. If the [animationDuration](#) property is set to **0**, which is the default value, the animation effect is disabled. If the animation is enabled, the component will behave in the following order.

1. The axis line, ticks, labels, and ranges will all be animated at the same time.
2. If available, pointers will be animated in the same way as [pointer animation](#).
3. If available, annotations will be animated.

The animation of the Linear Gauge is demonstrated in the following example.

APP.COMPONENT.TS

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

```

import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { AnnotationsService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ AnnotationsService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id='defaultContainer' background='transparent'
    style='display:block;'
      orientation='Horizontal' [annotations]='annotation' [axes]='axes'
    animationDuration="3000">
      </ejs-lineargauge>`,
})
export class AppComponent {
  public axes?: Object[];
  public annotation?: Object;
  ngOnInit(): void {
    this.axes = [
      {
        pointers: [
          {
            value: 10,
            height: 15,
            width: 15,
            placement: 'Near',
            offset: -40,
            markerType: 'Triangle',
          },
        ],
        ranges: [
          {
            start: 0,
            end: 50,
            startWidth: 10,
            endWidth: 10,
            color: '#F45656',
            offset: 35,
          },
        ],
        majorTicks: {
          interval: 10,
          height: 20,
          color: '#9E9E9E',
        },
        minorTicks: {
          interval: 2,
          height: 10,
          color: '#9E9E9E',
        },
        labelStyle: {
          offset: 48,
          font: { fontFamily: 'inherit' },
        },
      },
    ],
  }
}

```

```

    },
  ];
  this.annotation = [
    {
      content:
        '<div style="width: 70px;margin-left:-3%;margin-top: 42%;font-size: 16px;">10 MPH</div>',
      axisIndex: 0,
      axisValue: 10,
      x: 10,
      y: -70,
      zIndex: '1',
    },
  ];
}
}

```

MAIN.TS

```

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

```

Only the pointer of the Linear Gauge can be animated individually, not the axis lines, ticks, labels, ranges, and annotations. You can refer this [link](#) to enable only pointer animation.

User interaction in Angular Linear gauge component

<!-- markdownlint-disable MD036 -->

Tooltip

<!-- markdownlint-disable MD036 -->

Linear Gauge displays the details about a pointer value through [tooltip](#), when the mouse hovers over the pointer. To enable the tooltip, set [enable](#) property as **true** and inject the **GaugeTooltipService** in the **providers**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id='tooltipContainer' style='display:block;'
    [tooltip]='tooltip'>
      <e-axes>

```

```

        <e-axis>
            <e-pointers>
                <e-pointer value=80></e-pointer>
            </e-pointers>
        </e-axis>
    </e-axes>
</ejs-lineargauge>`
    })
    export class AppComponent implements OnInit {
        public tooltip?:Object;
        ngOnInit(): void {
            this.tooltip = {
                enable: true
            };
        }
    }
}

```

MAIN.TS

```

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

```

<!-- markdownlint-disable MD013 -->

Tooltip format

<!-- markdownlint-disable MD013 -->

Tooltip in the Linear Gauge control can be formatted using the [format](#) property in [tooltip](#). It is used to render the tooltip in certain format or to add a user-defined unit in the tooltip. By default, the tooltip shows the pointer value only. In addition to that, more information can be added in the tooltip. For example, the format **{value}km** shows pointer value with kilometer unit in the tooltip.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        LinearGaugeModule
    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
        <ejs-lineargauge id="gauge-container" style='display:block;'
        [tooltip]='tooltip'>
            <e-axes>
                <e-axis>
                    <e-pointers>
                        <e-pointer value=80></e-pointer>
                    </e-pointers>
            </e-axes>
        </ejs-lineargauge>`
})

```

```

        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })
  export class AppComponent implements OnInit {
    public tooltip?:Object;
    ngOnInit(): void {
      this.tooltip = {
        enable: true,
        format: '{value} km'
      };
    }
  }
}

```

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

The HTML element can be rendered in the tooltip of the Linear Gauge using the [template](#) property in [tooltip](#). The `#{value}` can be used as placeholders in the HTML element to display the pointer values of the corresponding axis.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" style='display:block;'
    [tooltip]='tooltip'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=80></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public tooltip?:Object;
  ngOnInit(): void {

```

```

        this.tooltip = {
            enable: true,
            //tooltip template for Linear gauge
            template: '<div>Pointer: 80 </div>'
        };
    }
}

```

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 the tooltip

The tooltip can be customized using the following properties in [tooltip](#).

- [fill](#) - To fill the color for tooltip.
- [enableAnimation](#) - To enable or disable the tooltip animation.
- [border](#) - To set the border color and width of the tooltip.
- [textStyle](#) - To customize the style of the text in tooltip.
- [showAtMousePosition](#) - To show the tooltip at the mouse position.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge';
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge';
import { Component, OnInit } from '@angular/core';
@Component({
    imports: [
        LinearGaugeModule
    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
        <ejs-lineargauge id="gauge-container" style="display:block;"
        [tooltip]='tooltip'>
            <e-axes>
                <e-axis>
                    <e-pointers>
                        <e-pointer value=80></e-pointer>
                    </e-pointers>
                </e-axis>
            </e-axes>
        </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
    public tooltip?:Object;
    ngOnInit(): void {
        this.tooltip = {

```

```

        enable: true,
        fill: '#e5bcbcb',
        border: {
            color: '#d80000',
            width: 2
        }
    };
}
}
}

```

MAIN.TS

```

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

```

Positioning the tooltip

The tooltip is positioned at the **End** of the pointer. To change the position of the tooltip at the start, or center of the pointer, set the [position](#) property to **Start** or **Center**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" style='display:block;'
    [tooltip]='tooltip'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=50 type="Bar" color="blue"></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public tooltip?:Object;
  ngOnInit(): void {
    this.tooltip = {
      enable: true,
      position: "Center"
    };
  }
}

```



```
}

```

MAIN.TS

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

```

Pointer Drag

To drag either marker or bar pointer to the desired axis value, set the [enableDrag](#) property as **true** in the [pointer](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-lineargauge id="gauge-container" style='display:block;'
height='350'>
    <e-axes>
      <e-axis>
        <e-pointers>
          <e-pointer value=80 [enableDrag]=true></e-pointer>
        </e-pointers>
      </e-axis>
    </e-axes>
  </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

```

Print and export in Angular Linear gauge component**Print**

```
<!-- markdownlint-disable MD013 -->

```

The rendered Linear Gauge can be printed directly from the browser by calling the [print](#) method. To use the print functionality, set the [allowPrint](#) property as **true** and inject the **PrintService** in the **providers**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { PrintService, LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [allowPrint]=true #gauge>
    </ejs-lineargauge><div> <button id='print'
  (click)='print()' >Print</button></div>`,
  providers: [PrintService]
})
export class AppComponent {
  @ViewChild('gauge')
  public gaugeObj: LinearGaugeComponent | any;
  print() {
    this.gaugeObj.print();
  };
}
```

MAIN.TS

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

Export

Image Export

<!-- markdownlint-disable MD013 -->

To use the image export functionality, set the [allowImageExport](#) property as **true** and inject the **ImageExportService** in the **providers**. The rendered Linear Gauge can be exported as an image using the [export](#) method. This method requires two parameters: export type and file name. The Linear Gauge can be exported as an image with the following formats.

- JPEG
- PNG
- SVG

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { ImageExportService, LinearGaugeComponent } from '@syncfusion/ej2-
angular-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [allowImageExport]=true #gauge>
    </ejs-lineargauge><div><button id='export'
    (click)='export()'>Export</button></div>`,
  providers: [ImageExportService]
})
export class AppComponent {
  @ViewChild('gauge')
  public gaugeObj: LinearGaugeComponent | any;
  public export() {
    this.gaugeObj.export('PNG', 'Gauge');
  };
}
```

MAIN.TS

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

PDF Export

To use the PDF export functionality, set the [allowPdfExport](#) property as **true** and inject the **PdfExportService** in the **providers**. The rendered Linear Gauge can be exported as PDF using the [export](#) method. The [export](#) method requires three parameters: file type, file name and orientation of the PDF document. The orientation of the PDF document can be set as **Portrait** or **Landscape**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { PdfExportService, LinearGaugeComponent } from '@syncfusion/ej2-
angular-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
```

```

    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
      <ejs-lineargauge id="gauge-container" [allowPdfExport]=true #gauge>
      </ejs-lineargauge><div> <button id='export'
      (click)='export()'>Export</button></div>`,
    providers: [PdfExportService]
  })
  export class AppComponent {
    @ViewChild('gauge')
    public gaugeObj: LinearGaugeComponent | any;
    public export(){
      this.gaugeObj.export('PDF', 'Gauge', 0);
    };
  }

```

MAIN.TS

```

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

```

Exporting Linear Gauge as base64 string of the file

The Linear Gauge can be exported as base64 string for the JPEG, PNG and PDF formats. The rendered Linear Gauge can be exported as base64 string of the exported image or PDF document used in the [export](#) method. The arguments that are required for this method is export type, file name, orientation of the exported PDF document and **allowDownload** boolean value that is set as **false** to return base64 string. The value for the orientation of the exported PDF document is set as **null** for image export and **Portrait** or **Landscape** for the PDF document.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { ImageExportService, LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [allowImageExport]=true #gauge>
    </ejs-lineargauge><div> <button id='export'
    (click)='export()'>Export</button></div>`,
  providers: [ImageExportService]
})

```

```
export class AppComponent {
  @ViewChild('gauge')
  public gaugeObj: LinearGaugeComponent | any;
  public export() {
    const promise = this.gaugeObj.export('PNG', 'Gauge', null, false);
    promise.then((data: string) => {
      document.writeln(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 exporting of the Linear Gauge as base64 string is not applicable for the SVG format.

Appearance in Angular Linear gauge component

<!-- markdownlint-disable MD013 -->

Customizing the Linear Gauge area

The following properties are available in the [ejs-lineargauge](#) to customize the Linear Gauge area.

- [background](#) - Applies the background color for the Linear gauge.
- [border](#) - To customize the color and width of the border in Linear Gauge.
- [margin](#) - To customize the margins of the Linear Gauge.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" background='skyblue'
    [border]='border' [margin]='margin'>
    </ejs-lineargauge>`
})
export class AppComponent {
  public border?: Object;
  public margin?: Object;
  ngOnInit(): void {
    this.border = { color: "#FF0000", width: 2 };
  }
}
```

```

    this.margin = { left: 20, top: 20, right: 20, bottom: 20}
  }
}

```

MAIN.TS

```

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

```

Setting up the Linear Gauge title

The title for the Linear Gauge can be set using [title](#) property in [ejs-lineargauge](#). Its appearance can be customized using the [titleStyle](#) with the below properties.

- [color](#) - Specifies the text color of the title.
- [fontFamily](#) - Specifies the font family of the title.
- [fontStyle](#) - Specifies the font style of the title.
- [fontWeight](#) - Specifies the font weight of the title.
- [opacity](#) - Specifies the opacity of the title.
- [size](#) - Specifies the font size of the title.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" title='Speedometer'
    [titleStyle]='titleStyle'>
    </ejs-lineargauge>`
})
export class AppComponent {
  public titleStyle?: Object;
  ngOnInit(): void {
    this.titleStyle = {
      fontFamily: "Arial",
      fontStyle: 'italic',
      fontWeight: 'regular',
      color: "#E27F2D",
      size: '23px'
    };
  }
}

```

MAIN.TS

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

Customizing the Linear Gauge container

The area used to render the ranges and pointers at the center position of the gauge is called container. The following types of container to be applicable for Linear Gauge.

- Normal
- Rounded Rectangle
- Thermometer

The type of the container can be modified by using the [type](#) property in [container](#). The container can be customized by using the following properties in [container](#).

- [offset](#) - To place the container with the specified distance from the axis of the Linear Gauge.
- [width](#) - To set the thickness of the container.
- [height](#) - To set the length of the container.
- [backgroundColor](#) - To set the background color of the container.
- [border](#) - To set the color and width for the border of the container.

Normal

The **Normal** type will render the container as a rectangle and this is the default container type.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer type="Bar" value=50>
            </e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </e-axes>
```

```

    </ejs-lineargauge>`
  })
  export class AppComponent implements OnInit {
    public container?: Object;
    ngOnInit(): void {
      this.container = {
        width:30
      };
    }
  }
}

```

MAIN.TS

```

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

```

Rounded Rectangle

The **RoundedRectangle** type will render the container as a rectangle with rounded corner radius. The rounded corner radius of the container can be customized using the [roundedCornerRadius](#) property in [container](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer type="Bar" value=50>
          </e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public container?: string | any;
  ngOnInit(): void {
    this.container = {
      width:30,
      type: "RoundedRectangle"
    }
  }
}

```



```
    };
  }
}
```

MAIN.TS

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

Thermometer

The **Thermometer** type will render the container similar to the appearance of thermometer.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [container]='container'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer type="Bar" value=50>
          </e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public container?: Object;
  ngOnInit(): void {
    this.container = {
      width:30,
      type: "Thermometer"
    };
  }
}
```

MAIN.TS

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

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

Fitting the Linear Gauge to the control

The Linear Gauge component is rendered with margin by default. To remove the margin around the Linear Gauge, the [allowMargin](#) property in [ejs-lineargauge](#) is set as **false**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" [margin]='margin'
    [allowMargin]='isMargin' orientation="Horizontal" background="skyblue"
    [border]='border'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer type="Bar" value=50>
          </e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent implements OnInit {
  public border?: Object;
  public margin?: Object;
  public isMargin?: boolean;
  ngOnInit(): void {
    this.isMargin = false;
    this.margin = {
      left: 0,
      right: 0,
      top: 0,
      bottom: 0
    },
    this.border = {
      width: 3,
      color: "red"
    };
  }
}
```

MAIN.TS

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

To use this feature, set the [allowMargin](#) property to **false**, the [width](#) property to **100%** and the properties of [margin](#) to **0**.

Accessibility in Angular Linear Gauge component

The Linear Gauge 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 Linear Gauge 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

The Linear Gauge component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Linear Gauge component:

| Attributes | Purpose |

| --- | --- |

| **role=region** | It is specified in the title and pointer. The pointer supports the interactive drag-and-drop function to update the pointer value. |

| **aria-label** | Provides an accessible name for the title, axis labels, text pointer and annotation. |

Screen reading in Linear Gauge

Accessibility in the Linear Gauge component ensures that all users, regardless of ability or disability, can use screen reading. The following Linear Gauge elements will be read aloud using screen reading software, such as Narrator for Windows.

| Elements | Description |

| --- | --- |

| Title | Reads the title of the Linear Gauge. |

| Axis labels | Reads the axis labels of the Linear Gauge. |

| Text pointer | Reads the text content shown as a pointer in Linear Gauge. |

| Annotation | Reads the content specified in the annotation. |

Ensuring accessibility

The Linear Gauge component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Linear Gauge component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Linear Gauge component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Internationalization in Angular Linear gauge component

Globalization is the process of designing and developing a component that works in different cultures. Internationalization is used to globalize the number content in Linear Gauge component using [format](#) property in [ejs-lineargauge](#). It has static text on some features such as

- Axis label
- Tooltip

The static text on above features can be changed to any culture such as Arabic, Deutsch and French. To know more about the globalization in Angular components, refer [here](#).

Numeric Format

The text in axis labels and tooltip can be displayed in the numeric format such as currency, percentage and so on. To know more about the numeric formats in axis labels, refer [here](#). In the below example, the axis label is displayed in the currency format.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" format="c">
    </ejs-lineargauge>`
})
export class AppComponent {
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Events in Angular Linear gauge component

This section describes the Linear Gauge component's event that gets triggered when corresponding operations are performed.

animationComplete

When the pointer animation is completed, the [animationComplete](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IAnimationCompleteEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
```

```

    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
      <ejs-lineargauge id="gauge-container"
      (animationComplete)="animationComplete($event)">
        <e-axes>
          <e-axis>
            <e-pointers>
              <e-pointer value=10></e-pointer>
            </e-pointers>
          </e-axis>
        </e-axes>
      </ejs-lineargauge>`
  ))
  export class AppComponent {
    animationComplete(args: IAnimationCompleteEventArgs) {
    };
  }
}

```

MAIN.TS

```

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

```

annotationRender

Before the annotation is rendered in the Linear Gauge, the [annotationRender](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { AnnotationsService, LinearGaugeComponent } from '@syncfusion/ej2-
angular-lineargauge';
import { IAnnotationRenderEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (annotationRender)="annotationRender($event)">
      <e-annotations>
        <e-annotation zIndex='1' content='<div
        id="first"><h1>Gauge</h1></div>' axisValue=0></e-annotation>

```

```

        </e-annotations>
    </ejs-lineargauge>`,
    providers: [AnnotationsService]
  })
  export class AppComponent {
    annotationRender(args: IAnnotationRenderEventArgs) {
    };
  }

```

MAIN.TS

```

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

```

axisLabelRender

Before each axis label is rendered in the Linear Gauge, the [axisLabelRender](#) event is fired. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IAxisLabelRenderEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (axisLabelRender)= 'axisLabelRender($event)' '>
    </ejs-lineargauge>`
})
export class AppComponent {
  axisLabelRender(args: IAxisLabelRenderEventArgs) {
  };
}

```

MAIN.TS

```

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

```

beforePrint

The [beforePrint](#) event is fired before the print begins. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { PrintService, LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IPrintEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <div><button id='print' (click)='print()'>Print</button></div>
    <ejs-lineargauge id="gauge-container" [allowPrint]=true
    (beforePrint)='beforePrint($event)' #gauge>
    </ejs-lineargauge>`,
  providers: [PrintService]
})
export class AppComponent {
  @ViewChild('gauge')
  public gaugeObj: LinearGaugeComponent | any;
  print() {
    this.gaugeObj.print();
  };
  beforePrint(args: IPrintEventArgs) {
  };
}
```

MAIN.TS

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

dragEnd

The [dragEnd](#) event will be fired before the pointer drag is completed. To know more about the argument of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
```



```

import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IPointerDragEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge style="display:block" id="gauge-container"
    (dragEnd)='dragEnd($event)'>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer enableDrag=true></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
})
export class AppComponent {
  dragEnd(args: IPointerDragEventArgs) {
  }
}

```

MAIN.TS

```

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

```

dragMove

The [dragMove](#) event will be fired when the pointer is dragged. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge';
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge';
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IPointerDragEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `

```

```

    <ejs-lineargauge style="display:block" id="gauge-container"
    (dragMove)='dragMove($event) '>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer enableDrag=true></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  ))
export class AppComponent {
  dragMove(args: IPointerDragEventArgs) {
  }
}

```

MAIN.TS

```

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

```

dragStart

When the pointer drag begins, the [dragStart](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IPointerDragEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge style="display:block" id="gauge-container"
    (dragStart)='dragStart($event) '>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer enableDrag=true></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>`
  })

```

```
export class AppComponent {
    dragStart(args: IPointerDragEventArgs) {
    }
}
```

MAIN.TS

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

gaugeMouseDown

When mouse is pressed down on the gauge, the [gaugeMouseDown](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IMouseEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (gaugeMouseDown)="gaugeMouseDown($event)" >
    </ejs-lineargauge>`
})
export class AppComponent {
    gaugeMouseDown(args: IMouseEventArgs) {
    }
}
```

MAIN.TS

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

gaugeMouseLeave

When mouse pointer moves over the gauge, the [gaugemouseleave](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IMouseEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (gaugeMouseLeave)="gaugeMouseLeave($event)" >
    </ejs-lineargauge>`
})
export class AppComponent {
  gaugeMouseLeave(args: IMouseEventArgs) {
  }
}
```

MAIN.TS

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

gaugeMouseMove

When mouse pointer leaves the gauge, the [gaugeMouseMove](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IMouseEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (gaugeMouseMove)="gaugeMouseMove($event)" >
    </ejs-lineargauge>`
})
export class AppComponent {
  gaugeMouseMove(args: IMouseEventArgs) {
  }
}
```

```

    })
    export class AppComponent {
        gaugeMouseMove(args: IMouseEventArgs) {
        }
    }
}

```

MAIN.TS

```

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

```

gaugeMouseUp

When the mouse pointer is released over the Linear Gauge, the [gaugeMouseUp](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IMouseEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (gaugeMouseUp)="gaugeMouseUp($event)" >
    </ejs-lineargauge>`
})
export class AppComponent {
    gaugeMouseUp(args: IMouseEventArgs) {
    }
}

```

MAIN.TS

```

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

```

load

Before the Linear Gauge is loaded, the [load](#) event is fired. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { ILoadEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" (load)='load($event)'>
    </ejs-lineargauge>`
})
export class AppComponent {
  load(args: ILoadEventArgs) {
  }
}
```

MAIN.TS

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

loaded

After the Linear Gauge has been loaded, the [loaded](#) event will be triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { ILoadedEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" (loaded)='loaded($event)'>
    </ejs-lineargauge>`
})
```

```
export class AppComponent {
    loaded(args: ILoadedEventArgs) {
    }
}
```

MAIN.TS

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

resized

After the window resizing, the [resized](#) event is triggered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IResizeEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container" (resized)='resized($event)'>
    </ejs-lineargauge>`
})
export class AppComponent {
  resized(args: IResizeEventArgs) {
  }
}
```

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 fired before the tooltip is rendered. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { ITooltipRenderEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (tooltipRender)='tooltipRender($event)'>
    </ejs-lineargauge>`
})
export class AppComponent {
  tooltipRender(args: ITooltipRenderEventArgs) {
  }
}
```

MAIN.TS

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

valueChange

The [valueChange](#) event is triggered when the pointer is dragged from one value to another. To know more about the arguments of this event, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
import { IValueChangeEventArgs } from '@syncfusion/ej2-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-lineargauge id="gauge-container"
    (valueChange)='valueChange($event)'>
    <e-axes>
```



```

        <e-axis>
            <e-pointers>
                <e-pointer value=40 enableDrag=true></e-pointer>
            </e-pointers>
        </e-axis>
    </e-axes>
</ejs-lineargauge>`
    ))
    export class AppComponent {
        valueChange(args: IValueChangeEventArgs) {
        }
    }
}

```

MAIN.TS

```

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

```

Methods in Angular Linear gauge component

The following methods are available in the Linear Gauge component.

setPointerValue

To change the pointer value dynamically, use the [setPointerValue](#) method in the Linear Gauge component. The following are the arguments for this method.

Argument name	Description
axisIndex	Specifies the index of the axis in which the pointer value is to be updated.
pointerIndex	Specifies the index of the pointer to be updated.
pointerValue	Specifies the value of the pointer to be updated.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
@Component({
    imports: [
        LinearGaugeModule
    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
        <div> <button id='btn' (click)='clicked()'>Click</button></div>
        <ejs-lineargauge id="gauge-container" #gauge>
            <e-axes>
                <e-axis>

```

```

        <e-pointers>
            <e-pointer value=80></e-pointer>
        </e-pointers>
    </e-axis>
</e-axes>
</ejs-lineargauge>`
    })
    export class AppComponent {
        @ViewChild('gauge')
        public gaugeObj: LinearGaugeComponent | any;
        clicked() {
            this.gaugeObj.setPointerValue(0, 0, 30);
        };
    }
}

```

MAIN.TS

```

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

```

setAnnotationValue

To change the annotation content dynamically, use the [setAnnotationValue](#) method in the Linear Gauge component. The following are the arguments for this method.

Argument name	Description
-----	-----
annotationIndex	Specifies the index number of the annotation to be updated.
content	Specifies the text for the annotation to be updated.
axisValue	Specifies the value of the axis where the annotation is to be placed.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { AnnotationsService, LinearGaugeComponent } from '@syncfusion/ej2-
angular-lineargauge';
@Component({
    imports: [
        LinearGaugeModule
    ],
    providers: [ GaugeTooltipService ],
    standalone: true,
    selector: 'app-container',
    template: `
        <div> <button id='btn' (click)='clicked()'>Click</button></div>
        <ejs-lineargauge id="gauge-container" #gauge>
            <e-annotations>

```

```

        <e-annotation zIndex='1' content='10' axisValue=0></e-
annotation>
    </e-annotations>
    <e-axes>
        <e-axis>
            <e-pointers>
                <e-pointer value=10></e-pointer>
            </e-pointers>
        </e-axis>
    </e-axes>
</ejs-lineargauge>`,
providers: [AnnotationsService]
})
export class AppComponent {
    @ViewChild('gauge')
    public gaugeObj: LinearGaugeComponent | any;
    clicked() {
        this.gaugeObj.setAnnotationValue(0, '50', 50);
    };
}

```

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 [refresh](#) method can be used to change the state of the component and render it again.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { LinearGaugeModule } from '@syncfusion/ej2-angular-lineargauge'
import { GaugeTooltipService } from '@syncfusion/ej2-angular-lineargauge'
import { Component, ViewChild } from '@angular/core';
import { LinearGaugeComponent } from '@syncfusion/ej2-angular-lineargauge';
@Component({
  imports: [
    LinearGaugeModule
  ],
  providers: [ GaugeTooltipService ],
  standalone: true,
  selector: 'app-container',
  template: `
    <div> <button id='btn' (click)='clicked()'>Click</button></div>
    <ejs-lineargauge id="gauge-container" #gauge>
      <e-axes>
        <e-axis>
          <e-pointers>
            <e-pointer value=10></e-pointer>
          </e-pointers>
        </e-axis>
      </e-axes>
    </ejs-lineargauge>
  `
})

```

```

    </ejs-lineargauge>`
  })
  export class AppComponent {
    @ViewChild('gauge')
    public gaugeObj : LinearGaugeComponent | any;
    clicked() {
      this.gaugeObj.refresh();
    };
  }

```

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 Linear gauge component

This article describes the API migration process of Accordion component from Essential JS 1 to Essential JS 2.

Linear gauge dimensions

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Height | **Property:** *height*
<ej-lineargauge height="150"> </ej-lineargauge> | **Property:** *height*
<ejs-lineargauge height="150px"> </ejs-lineargauge> |

| Width | **Property:** *width*
<ejs-lineargauge width="200"> </ej-lineargauge> | **Property:** *width*
<ej-lineargauge width="200px"> </ejs-lineargauge> |

| Height(In Percentage) | Not Applicable | **Property:** *height*
<ejs-lineargauge height='70%'> </ejs-lineargauge> |

| Width(In Percentage) | Not Applicable | **Property:** *width*
<ejs-lineargauge width='100%'> </ejs-lineargauge> |

Line customizaton

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Height | **Property:** *scales.length*
<ej-lineargauge><e-scales><e-scale length 300> </e-scale> </e-scales> </ej-lineargauge> | **Property:** *axes.line.height*
<ejs-lineargauge><e-axes><e-axis [line]='Line'> </e-axis> </e-axes> </ej-lineargauge>

public Line:Object = { height: "150"} |

| Width | **Property:** *scales.width*
<ej-lineargauge><e-scales><e-scale width = "300"> </e-scale> </e-scales> </ej-lineargauge> | **Property:** *axes.line.width*
<ejs-lineargauge><e-axes><e-axis [line]='Line'> </e-axis> </e-axes> </ej-lineargauge>

public Line:Object = { width: 150} |

|Color| **Property:** *scales.backgroundColor*
`<ej-lineargauge><e-scales><e-scale
 backgroundColor="red"></e-scale></e-scales> </ej-lineargauge>` | **Property:**
axes.line.color
`<ej-lineargauge><e-axes><e-axis [line]='Line'></e-axis></e-axes>
 </ej-lineargauge>` `

public Line:Object = { color:"red"}|`

|Offset| Not Applicable | **Property:** *axes.line.offset*
`<ej-lineargauge><e-axes><e-axis
 [line]='Line'></e-axis></e-axes> </ej-lineargauge>` `

public Line:Object = { offset : 2}|`

|Opacity| **Property:** *scales.opacity*
`<ej-lineargauge><e-scales><e-scale opacity= "0.2"
 ></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

|DashArray| Not Applicable | **Property:** *axes.line.dashArray*
`<ej-lineargauge><e-axes><e-axis [line]='Line'></e-axis></e-axes> </ej-lineargauge>` `

public Line:Object = {
 dashArray : '1'}|`

Ticks

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

|Type of Ticks| **Property:** *scales.ticks.type*
`<ej-lineargauge><e-scales><e-scale><e-
 ticks><e-tick type="majorinterval"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.majorTicks.height*
`<ej-lineargauge><e-axes><e-axis
 [majorTicks]='majorTicks'></e-axis></e-axes> </ej-lineargauge>` `

public majorTicks:
 Object= { }|`

|Height of Major Ticks| **Property:** *scales.ticks.height*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="majorinterval" height= "8"></e-tick></e-ticks></e-scale></e-
 scales> </ej-lineargauge>` | **Property:** *axes.majorTicks.height*
`<ej-lineargauge><e-
 axes><e-axis [majorTicks]='majorTicks'></e-axis></e-axes> </ej-lineargauge>` `

public
 majorTicks: Object= { height: 8 }|`

|Width of Major Ticks| **Property:** *scales.ticks.width*
`<ej-lineargauge><e-scales><e-
 scale><e-ticks><e-tick type="majorinterval" width= "5"></e-tick></e-ticks></e-scale></e-
 scales> </ej-lineargauge>` | **Property:** *axes.majorTicks.width*
`<ej-lineargauge><e-
 axes><e-axis [majorTicks]='majorTicks'></e-axis></e-axes> </ej-lineargauge>` `

public
 majorTicks: Object= { width: 5 }|`

|Color of Major Ticks| **Property:** *scales.ticks.color*
`<ej-lineargauge><e-scales><e-scale><e-
 ticks><e-tick type="majorinterval" color='blue'></e-tick></e-ticks></e-scale></e-scales> </ej-
 lineargauge>` | **Property:** *axes.majorTicks.color*
`<ej-lineargauge><e-axes><e-axis
 [majorTicks]='majorTicks'></e-axis></e-axes> </ej-lineargauge>` `

public majorTicks:
 Object= { color:'blue' }|`

|Offset for Major Ticks| **Property:** *scales.ticks.distanceFromScale*
`<ej-lineargauge><e-
 scales><e-scale><e-ticks><e-tick type="majorinterval" [distanceFromScale]="{ x: 5, y: 5 }"></e-
 tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.majorTicks.offset*
`<ej-lineargauge><e-axes><e-axis [majorTicks]='majorTicks'></e-axis></e-axes> </ej-
 lineargauge>` `

public majorTicks: Object= { offset : 1 }|`

|Interval of Major Ticks| **Property:** *scales.majorIntervalValue*
`<ej-lineargauge><e-scales><e-scale majorIntervalValue= "15"></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.majorTicks.interval*
`<ejs-lineargauge><e-axes><e-axis [majorTicks]='majorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public majorTicks: Object= { interval : 15 }` |

|Angle of Major Ticks| **Property:** *scales.ticks.angle*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="majorinterval" angle= "30"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

|Opcity of Major Ticks| **Property:** *scales.ticks.opacity*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="majorinterval" opacity= "0.5"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

|Height of Minor Ticks| **Property:** *scales.ticks.height*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" height= "8"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.minorTicks.height*
`<ejs-lineargauge><e-axes><e-axis [minorTicks]='minorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public minorTicks: Object= { height: 8 }` |

|Width of Minor Ticks| **Property:** *scales.ticks.width*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" width= "5"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.minorTicks.width*
`<ejs-lineargauge><e-axes><e-axis [minorTicks]='minorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public minorTicks: Object= { width: 5 }` |

|Color of Minor Ticks| **Property:** *scales.ticks.color*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" color='blue'></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.minorTicks.color*
`<ejs-lineargauge><e-axes><e-axis [minorTicks]='minorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public minorTicks: Object= { color:'blue' }` |

|Offset for Major Ticks| **Property:** *scales.ticks.distanceFromScale*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" [distanceFromScale]="distanceFromScale"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>`
`public distanceFromScale: Object= { x: 5, y: 5 }` | **Property:** *axes.minorTicks.offset*
`<ejs-lineargauge><e-axes><e-axis [minorTicks]='minorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public minorTicks: Object= { offset : 1 }` |

|Interval of Minor Ticks| **Property:** *scales.majorIntervalValue*
`<ej-lineargauge><e-scales><e-scale minorIntervalValue= "15"></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.minorTicks.interval*
`<ejs-lineargauge><e-axes><e-axis [minorTicks]='minorTicks'></e-axis></e-axes> </ejs-lineargauge>`
`public minorTicks: Object= { interval : 15 }` |

| Angle of Minor Ticks | **Property:** *scales.ticks.angle*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" angle= "30"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

| Opacity of Minor Ticks | **Property:** *scales.ticks.opacity*
`<ej-lineargauge><e-scales><e-scale><e-ticks><e-tick type="minorinterval" opacity= "0.5"></e-tick></e-ticks></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

Labels

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Angle | **Property:** *scales.labels.angle*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{ angle: 10}]` | Not Applicable |

| Offset | **Property:** *scales.labels.distanceFromScale*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{ distanceFromScale: { x: 0, y: 60 } }]` | **Property:** *axes.labelStyle.offset*
`<ejs-lineargauge><e-axes><e-axis [labelStyle]= "labelStyle"></e-axis></e-axes> </ejs-lineargauge>`
`public labelStyle: Object= { offset : 3 }` |

| Format | **Property:** *scales.labels.unitText*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{ unitText: "F"}]` | **Property:** *axes.labelStyle.format*
`<ejs-lineargauge><e-axes><e-axis [labelStyle]= "labelStyle"></e-axis></e-axes> </ejs-lineargauge>`
`public labelStyle: Object= { format: 'c' }` |

| Unit Text Placement | **Property:** *scales.labels.unitTextPlacement*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{ unitTextPlacement: "front"}]` | Not Applicable |

| Label Range Color | Not Applicable | **Property:** *axes.labelStyle.useRangeColor*
`<ejs-lineargauge><e-axes><e-axis [labelStyle]= "labelStyle"></e-axis></e-axes> </ejs-lineargauge>`
`public labelStyle: Object= { useRangeColor: true }` |

| Opacity | **Property:** *scales.labels.opacity*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{opacity: 0.5}]` | **Property:** *axes.labelStyle.font.opacity*
`<ejs-lineargauge><e-axes><e-axis [labelStyle]= "labelStyle"></e-axis></e-axes> </ejs-lineargauge>`
`public labelStyle: Object= { font: { opacity: 5 } }` |

| Label Text Color | **Property:** *scales.labels.textColor*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales> </ej-lineargauge>`
`public labels:Object = [{ textColor: "Red",}]` | **Property:** *axes.labelStyle.font.color*
`<ejs-lineargauge><e-axes><e-axis [labelStyle]= "labelStyle"></e-axis></e-axes> </ejs-lineargauge>`
`public labelStyle: Object= { font:{ color: 'red' } }` |

| Label Font Family | **Property:** *scales.labels.font.fontFamily*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales></ej-lineargauge>`
Property: *axes.labelStyle.font.fontFamily*
`<ej-lineargauge><e-axes><e-axis [labelStyle]="labelStyle"></e-axis></e-axes></ej-lineargauge>`
public labels: Object = { font: { fontFamily: 'Arial' } }

| Label Font Style | **Property:** *scales.labels.font.fontStyle*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales></ej-lineargauge>`
Property: *axes.labelStyle.font.fontStyle*
`<ej-lineargauge><e-axes><e-axis [labelStyle]="labelStyle"></e-axis></e-axes></ej-lineargauge>`
public labels: Object = { font: { fontStyle: 'Bold' } }

| Label Size | **Property:** *scales.labels.font.size*
`<ej-lineargauge><e-scales><e-scale [labels]="labels"></e-scale></e-scales></ej-lineargauge>`
Property: *axes.labelStyle.font.size*
`<ej-lineargauge><e-axes><e-axis [labelStyle]="labelStyle"></e-axis></e-axes></ej-lineargauge>`
public labelStyle: Object = { font: { size: "15px" } }

| Label Font Weight | Not Applicable | **Property:** *axes.labelStyle.font.fontWeight*
`<ej-lineargauge><e-axes><e-axis [labelStyle]="labelStyle"></e-axis></e-axes></ej-lineargauge>`
public labelStyle: Object = { font: { fontWeight: '4' } }

Axis

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Minimum Value | **Property:** *scales.minimum*
`<ej-lineargauge><e-scales><e-scale minimum="10"></e-scale></e-scales></ej-lineargauge>`
Property: *axes.minimum*
`<ej-lineargauge><e-axes><e-axis minimum="10"></e-axis></e-axes></ej-lineargauge>`

| Maximum Value | **Property:** *scales.maximum*
`<ej-lineargauge><e-scales><e-scale maximum="200"></e-scales></ej-lineargauge>`
Property: *axes.maximum*
`<ej-lineargauge><e-axes><e-axis maximum="200"></e-axis></e-axes></ej-lineargauge>`

| Inverted Position | Not Applicable | **Property:** *axes.isInversed*
`<ej-lineargauge><e-axes><e-axis [isInversed]='Direction'></e-axis></e-axes></ej-lineargauge>`

| Opposed Position | Not Applicable | **Property:** *axes.opposedPosition*
`<ej-lineargauge><e-axes><e-axis [opposedPosition]='position'></e-axis></e-axes></ej-lineargauge>`

Ranges

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Start Value | **Property:** *scales.ranges.startValue*
`<ej-lineargauge><e-scales><e-scale ranges><e-range startValue="20"></e-range></e-scales></ej-lineargauge>`
Property: *axes.ranges.start*
`<ej-lineargauge><e-axes><e-axis ranges><e-range start="20"></e-range></e-axes></ej-lineargauge>`

|End Value| **Property:** *scales.ranges.endValue*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range endValue= "20" ></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.end*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range end= "20"></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|Start Width| **Property:** *scales.ranges.startWidth*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range startWidth= "10" ></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.startWidth*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range startWidth= "10"></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|End Width| **Property:** *scales.ranges.endWidth*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range endWidth= "15"></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.endWidth*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range endWidth= "15"></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|Color| **Property:** *scales.ranges.backgroundColor*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range backgroundColor= "red" ></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.color*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range color= "red"></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|Offset| **Property:** *scales.ranges.distanceFromScale*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range distanceFromScale= "10" ></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.offset*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range offset= "10" ></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|Range Position| **Property:** *scales.ranges.placement*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range placement: "Near"></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | **Property:** *axes.ranges.position*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range position= 'Inside' ></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>` |

|Opacity| **Property:** *scales.ranges.opacity*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range opacity: "0.3"></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>` | Not Applicable |

|Border Customization| **Property:** *scales.ranges.border*
`<ej-lineargauge><e-scales><e-scale><e-ranges><e-range [border]="border"></e-range></e-ranges></e-scale></e-scales> </ej-lineargauge>`
 public border: Object= { color: "blue", width: 2 } | **Property:** *axes.ranges.border*
`<ejs-lineargauge><e-axes><e-axis ><e-ranges><e-range [border]="border"></e-range></e-ranges></e-axis></e-axes> </ejs-lineargauge>`
 public border: Object= { color: "blue", width: 2 } |

Bar Pointer

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Bar Pointer | **Property:** `scales.ranges.barPointers.value`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="20"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>` | **Property:** `axes.pointers.value`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer type="Bar", value="20"></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>` |

| Color of Bar Pointer | **Property:** `scales.ranges.barPointers.backgroundColor`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="20", backgroundColor="red"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>` | **Property:** `axes.pointers.color`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer type="Bar" value="20" color='red'></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>` |

| Offset of Bar Pointer | **Property:** `scales.ranges.barPointers.distanceFromScale`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="40" distanceFromScale="20"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>` | **Property:** `axes.pointers.offset`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer type="Bar" value="20" offset="20"></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>` |

| Opacity of Bar Pointer | **Property:** `scales.ranges.barPointers.opacity`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="40" opacity="0.5"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>` | **Property:** `axes.pointers.opacity`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer type="Bar" value="20" opacity="0.5"></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>` |

| Width of Bar Pointer | **Property:** `scales.ranges.barPointers.width`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="40" width="25"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>` | **Property:** `axes.pointers.width`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer type="Bar" value="20" width="25"></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>` |

| Gradients of Bar Pointer | **Property:** `scales.ranges.barPointers.gradients`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer value="40" gradients="gradients:"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>`
 public gradients: Object = {colorInfo:[{ colorStop : 0, color:"#FFFFFF"}] } |
 Not Applicable |

| Border of Bar Pointer | **Property:** `scales.ranges.barPointers.border`
`<ej-lineargauge><e-scales><e-scale><e-barpointers><e-barpointer border="border"></e-barpointer></e-barpointers></e-scale></e-scales></ej-lineargauge>`
 public border: Object= { color: "red", width: 2 } | **Property:** `axes.pointers.border`
`<ejs-lineargauge><e-axes><e-axis><e-pointers><e-pointer border="border"></e-pointer></e-pointers></e-axis></e-axes></ejs-lineargauge>`
 public border : Object={ color: 'red', width: 2.5 } |

| Animation of Bar Pointer | **Property:** `enableAnimation`
`<ej-lineargauge>enableAnimation="true" ></ej-lineargauge>` | **Property:** `axes.pointers.animationDuration`
`
</br>`

```
<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Bar' value="20"
animationDuration="2500"></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|
```

| Rounded Corner of Bar Pointer | Not Applicable | **Property:**

```
axes.pointers.roundedCornerRadius<br/><br/> <ejs-lineargauge><e-axes><e-axis ><e-pointers><e-
pointer type= 'Bar' value="20" roundedCornerRadius="15"></e-pointer></e-pointers></e-
axis></e-axes> </ejs-lineargauge>|
```

Marker Pointer

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| marker Pointer | **Property:** *scales.ranges.markerPointers.value*

 <ej-lineargauge><e-
scales><e-scale><e-markerpointers><e-markerpointer value="20" ></e-markerpointer></e-
markerpointers></e-scale></e-scales> </ej-lineargauge>| **Property:** *axes.pointers.value*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type: "marker", value="20" ></e-
pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

| Color of marker Pointer | **Property:** *scales.ranges.markerPointers.backgroundColor*

 <ej-
lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value= "20"
backgroundColor="red"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-
lineargauge>| **Property:** *axes.pointers.color*

 <ejs-lineargauge><e-axes><e-axis ><e-
pointers><e-pointer type= 'Marker' value= "20" color='red'></e-pointer></e-pointers></e-
axis></e-axes> </ejs-lineargauge>|

| Offset of marker Pointer | **Property:** *scales.ranges.markerPointers.distanceFromScale*

 <ej-
lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value="40",
distanceFromScale="20" ></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-
lineargauge>| **Property:** *axes.pointers.offset*

 <ejs-lineargauge><e-axes><e-axis ><e-
pointers><e-pointer type= 'Marker' value="20" offset="20"></e-pointer></e-pointers></e-
axis></e-axes> </ejs-lineargauge>|

| Opacity of marker Pointer | **Property:** *scales.ranges.markerPointers.opacity*

 <ej-
lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value="40"
opacity="0.5"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-
lineargauge>| **Property:** *axes.pointers.opacity*

 <ejs-lineargauge><e-axes><e-axis ><e-
pointers><e-pointer type= 'Marker' value="20" opacity="0.5"></e-pointer></e-pointers></e-
axis></e-axes> </ejs-lineargauge>|

| Width of marker Pointer | **Property:** *scales.ranges.markerPointers.width*

 <ej-
lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value= "40" width="25"
></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-lineargauge>| **Property:**
axes.pointers.width

 <ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type=
'Marker' value="20" width="25"></e-pointer></e-pointers></e-axis></e-axes> </ejs-
lineargauge>|

| Gradients of marker Pointer | **Property:** *scales.ranges.markerPointers.gradients*

 <ej-
lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value= "40"

gradients="gradients:"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-lineargauge>

public gradients: Object =[{colorInfo:[{ colorStop : 0, color:"#FFFFFF"}]}]|
Not Applicable|

| Border of marker Pointer| **Property:** *scales.ranges.markerPointers.border*

<ej-lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer border="border"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-lineargauge>

public border: Object= { color: "red", width: 2 }| **Property:** *axes.pointers.border*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer border="border" ></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>

 public border : Object={ color: 'red', width: 2.5 }|

| Animation of marker Pointer| **Property:** *enableAnimation*

<ej-lineargauge enableAnimation="true" > </ej-lineargauge>| **Property:** *axes.pointers.animationDuration*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Marker' value="20" animationDuration="2500"></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

| Type of Marker Pointer| **Property:** *scales.ranges.markerPointers.type*

<ej-lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value="40" type= "Diamond"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-lineargauge>| **Property:** *axes.pointers.markerType*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Marker' value="20" markerType='Diamond'></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

| Placement of Marker Pointer| **Property:** *scales.ranges.markerPointers.placement*

<ej-lineargauge><e-scales><e-scale><e-markerpointers><e-markerpointer value="40" placement="Near"></e-markerpointer></e-markerpointers></e-scale></e-scales> </ej-lineargauge>| **Property:** *axes.pointers.placement*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Marker' value="20" placement= 'Center' ></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

| Drag of Marker Pointer| **Property:** *readOnly*

<ej-lineargauge readOnly= "false"> </ej-lineargauge>| **Property:** *axes.pointers.enableDrag*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Marker' value="20" enableDrag="true"></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

| Image Marker Pointer| Not Applicable| **Property:** *axes.pointers.imageUrl*

<ejs-lineargauge><e-axes><e-axis ><e-pointers><e-pointer type= 'Marker' value="20" imageUrl= './image.png' ></e-pointer></e-pointers></e-axis></e-axes> </ejs-lineargauge>|

Annotation

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Content| **Property:** *scales.customLabels.value*

<ej-lineargauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-lineargauge>

public customLabels: Object=[{value: "LinearGauge"}]| **Property:**

annotations.content

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation" } }|

| Horizontal Alignment | Not Applicable | **Property:** *annotations.horizontalAlignment*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", horizontalAlignment: 'Center' } }|

| Vertical Alignment | Not Applicable | **Property:** *annotations.verticalAlignment*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", verticalAlignment: 'Far' } }|

| Position of X | **Property:** *scales.customLabels.position.x*

 <ej-linear gauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-linear gauge>

public customLabels: Object={ { value: "LinearGauge", position: { x: 20 } } }|
Property: *annotations.x*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", x: 35 } }|

| Position of Y | **Property:** *scales.customLabels.position.y*

 <ej-linear gauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-linear gauge>

public customLabels: Object={ { value: "LinearGauge", position: { y: 30 } } }|
Property: *annotations.y*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", y: 40 } }|

| Z Index | Not Applicable | **Property:** *annotations.zIndex*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", zIndex: 1 } }|

| Axis Index | Not Applicable | **Property:** *annotations.axisIndex*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", axisIndex: 0 } }|

| Axis Value | Not Applicable | **Property:** *annotations.axisValue*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", axisValue: 35 } }|

| Font customization | **Property:** *scales.customLabels.font*

 <ej-linear gauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-linear gauge>

public customLabels: Object={ { value: "LinearGauge", font: { size: "30px" } } }| **Property:** *annotations.font*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", font: { size: '15px' } } }|

| Annotation Color | **Property:** *scales.customLabels.color*

 <ej-linear gauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-linear gauge>

public customLabels: Object={ { value: "LinearGauge", color: "yellow" } }|
Property: *annotations.font*

 <ejs-linear gauge [annotations]='annotation'></ejs-linear gauge>

public annotation:Object[]={ { content: "Annotation", font: { color: 'red' } } }|

|Opacity of Annotation| **Property:** *scales.customLabels.opacity*

 <ej-lineargauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-lineargauge>

public customLabels: Object=[{value: "LinearGauge", opacity: 0.5}]| **Property:** *annotations.font*

 <ejs-lineargauge [annotations]='annotation'></ejs-lineargauge>

public annotation:Object=[{ content: "Annotation", font: { opacity: 0.7 }}]|

|Position Type| **Property:** *scales.customLabels.positionType*

 <ej-lineargauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-lineargauge>

public customLabels: Object=[{value: "LinearGauge", positionType: "outer"}]| Not applicable|

|TextAngle of Annotation| **Property:** *scales.customLabels.textAngle*

 <ej-lineargauge><e-scales><e-scale [showCustomLabels]="true" [customLabels]="customLabels"></e-scale></e-scales> </ej-lineargauge>

public customLabels: Object=[{value: "LinearGauge", textAngle: 25 }]| Not applicable|

Tooltip

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

|Tooltip for Pointer| Not Applicable| **Property:** *tooltip.enable*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={enable: true}|

|Tooltip for Label| **Property:** *tooltip.showLabelTooltip*

 <ej-lineargauge [tooltip]="tooltip"></ej-lineargauge>

public tooltip:Object={ showLabelTooltip: true }| Not Applicable|

|Tooltip Format| Not Applicable| **Property:** *tooltip.format*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object=[{}]|

|Tooltip Color| Not Applicable| **Property:** *tooltip.fill*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={ enable: true, fill: 'gray'}|

|Tooltip Template| **Property:** *tooltip.templateID*

 <ej-lineargauge [tooltip]="tooltip"></ej-lineargauge>

public tooltip:Object={showLabelTooltip: true, showCustomLabelTooltip: true, templateID: 'Tooltip'}| **Property:** *tooltip.template*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={enable: true, template: 'Template'}|

|Tooltip Animation| Not Applicable| **Property:** *tooltip.enableAnimation*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={enable: true, enableAnimation: true}|

|Tooltip Border| Not Applicable| **Property:** *tooltip.border*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={enable: true, border: { width: 2, color: 'red' } }|

| Tooltip TextStyle| Not Applicable| **Property:** *tooltip.textStyle*

 <ejs-lineargauge [tooltip]='Tooltip'> </ejs-lineargauge>

public Tooltip: Object={enable: true, textStyle: { color: 'white', size: '10px' } }|

Appearance of Linear Gauge

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Background Color| **Property:** *backgroundColor*

 <ej-lineargauge backgroundColor= "Red"> </ej-lineargauge>| **Property:** *background*

 <ejs-lineargauge background= 'skyblue' ></ejs-lineargauge>|

| Border Color| **Property:** *borderColor*

 <ej-lineargauge borderColor= "Black" > </ej-lineargauge>| **Property:** *border.color*

<ejs-lineargauge [border]="border"></ejs-lineargauge>

public border: Object={color: 'red'}|

| Margin| Not Applicable| **Property:** *margin*

 <ejs-lineargauge [margin]="margin"></ejs-lineargauge>

public margin: Object={eft: 40, right: 40, top: 40, bottom: 40}|

| Orientation| **Property:** *orientation*

 <ej-lineargauge orientation= "Vertical"> </ej-lineargauge>| **Property:** *orientation*

 <ejs-lineargauge orientation= 'Horizontal'></ejs-lineargauge>|

| Locale| **Property:** *locale*

<ej-lineargauge locale= "en-US"> </ej-lineargauge>| **Property:** *locale*

 <ejs-lineargauge locale= 'en-US' ></ejs-lineargauge>|

| Theme| **Property:** *theme*

 <ej-lineargauge theme= 'Highcontrast' > </ej-lineargauge>| **Property:** *theme*

 <ejs-lineargauge theme= 'Highcontrast' ></ejs-lineargauge>|

| Gauge Title| Not Applicable| **Property:** *title*

 <ejs-lineargauge title= 'Linear Gauge'></ejs-lineargauge>|

Gauge Container type

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Container Type| **Property:** *scales.type*

<ej-lineargauge > <e-scales><e-scale type= 'thermometer'> </e-scales></e-scale> </ej-lineargauge>| **Property:** *container.type*

<ejs-lineargauge [container]='Container'></ejs-lineargauge>

public Container: Object={type: 'Thermometer'}|

| Container Height| Not Applicable| **Property:** *container.height*

 <ejs-lineargauge [container]='Container'></ejs-lineargauge>

public Container: Object={height: 20}|

| Container Width| Not Applicable| **Property:** *container.width*

<ejs-lineargauge [container]='Container'></ejs-lineargauge>

public Container: Object={ width: 10}|

| Container Offset| Not Applicable| **Property:** *container.offset*

 <ejs-lineargauge [container]='Container'></ejs-lineargauge>

public Container: Object={ offset: 5}|

Events

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Animation Complete Event| Not Applicable| **Event:** *animationComplete*

 <ejs-lineargauge (animationComplete)=‘animationComplete(\$event)’> </ejs-lineargauge>

public animationComplete(args: IAnimationCompleteEventArgs)=>{}|

| Annotation Render Event| **Event:** *drawCustomLabel*

 <ej-lineargauge (drawCustomLabel)=‘DrawCustomLabel(\$event)’> </ej-lineargauge>

DrawCustomLabel(sender){}| **Event:** *annotationRender*

 <ejs-lineargauge (load)=‘load(\$event)’> </ejs-lineargauge>

public load(args: ILoadedEventArgs)=>{}|

| AxisLabel Render Event| **Event:** *drawLabels*

 <ej-lineargauge (drawLabels)=‘drawLabels(\$event)’> </ej-lineargauge>

drawLabels(sender){}| **Event:** *axisLabelRender*

 <ejs-lineargauge (axisLabelRender)=‘axisLabelRender(\$event)’> </ejs-lineargauge>

public axisLabelRender(args: IAxisLabelRenderEventArgs)=>{}|

| Load Event| **Event:** *load*

 <ej-lineargauge (load)=‘load(\$event)’> </ej-lineargauge>

DrawRange(sender){}| **Event:** *load*

 <ejs-lineargauge (load)=‘load(\$event)’> </ejs-lineargauge>

public load(args: ILoadEventArgs)=>{}|

| Loaded Event| Not Applicable| **Event:** *loaded*

 <ejs-lineargauge (loaded)=‘loaded(\$event)’> </ejs-lineargauge>

public loaded(args: ILoadedEventArgs)=>{}|

| Resize Event| Not Applicable| **Event:** *resized*

 <ejs-lineargauge (resized)=‘resized(\$event)’> </ejs-lineargauge>

public resized(args: IResizeEventArgs)=>{}|

| Tooltip Render Event| Not Applicable| **Event:** *tooltipRender*

 <ejs-lineargauge (tooltipRender)=‘tooltipRender(\$event)’> </ejs-lineargauge>

public tooltipRender(args: ITooltipRenderEventArgs)=>{}|

| Value Change Event| Not Applicable| **Event:** *valueChange*

 <ejs-lineargauge (valueChange)=‘valueChange(\$event)’> </ejs-lineargauge>

public axisLabelRender(args: IValueChangeEventArgs)=>{}|

| Mouse Move Event| **Event:** *mouseClickMove*

 <ej-lineargauge (mouseClickMove)=‘MouseClickMove(\$event)’> </ej-lineargauge>

MouseClickMove(sender){}| **Event:** *gaugeMouseMove*

 <ejs-lineargauge (gaugeMouseMove)=‘gaugeMouseMove(\$event)’> </ejs-lineargauge>

public gaugeMouseMove(args: IMouseEventArgs)=>{}|

| Mouse Up Event| **Event:** *mouseClickUp*

 <ej-lineargauge (mouseClick)=‘MouseClick(\$event)’> </ej-lineargauge>

MouseClick(sender){}| **Event:** *gaugeMouseUp*

 <ejs-lineargauge (gaugeMouseUp)=‘gaugeMouseUp(\$event)’> </ejs-lineargauge>

public gaugeMouseUp(args: IMouseEventArgs)=>{}|

| Mouse Down Event| Not Applicable| **Event:** *gaugeMouseDown*

 <ejs-lineargauge (gaugeMouseDown)=‘gaugeMouseDown(\$event)’> </ejs-lineargauge>

public gaugeMouseDown(args: IMouseEventArgs)=>{}|

| Mouse Leave Event | Not Applicable | **Event:** *gaugeMouseLeave*

 <ejs-lineargauge (gaugeMouseLeave)= ' gaugeMouseLeave(\$event)'> </ejs-lineargauge>

public gaugeMouseLeave(args: IMouseEventArgs)=>{}|

| Mouse Click Event | **Event:** *mouseClick*

 <ej-lineargauge (mouseClick)= 'MouseClicked(\$event)'> </ej-lineargauge>

MouseClicked(sender){}| Not Applicable |

| Render Complete Event | **Event:** *renderComplete*

 <ej-lineargauge (renderComplete)= 'RenderComplete(\$event)'> </ej-lineargauge>

RenderComplete(sender){}| Not Applicable |

| Double Click Event | **Event:** *doubleClick*

 <ej-lineargauge (doubleClick)= 'DoubleClick(\$event)'> </ej-lineargauge>

DoubleClick(sender){}| Not Applicable |

| Right Click Event | **Event:** *rightClick*

 <ej-lineargauge (rightClick)= 'RightClick(\$event)'> </ej-lineargauge>

RightClick(sender){}| Not Applicable |

| BarPointers Event | **Event:** *drawBarPointers*

 <ej-lineargauge (drawBarPointers)= 'DrawBarPointers(\$event)'> </ej-lineargauge>

DrawBarPointers(sender){}| Not Applicable |

| Indicators Event | **Event:** *drawIndicators*

 <ej-lineargauge (drawIndicators)= 'DrawIndicators(\$event)'> </ej-lineargauge>

DrawIndicators(sender){}| Not Applicable |

| MarkerPointer Event | **Event:** *drawMarkerPointers*

 <ej-lineargauge (drawMarkerPointers)= 'DrawMarkerPointers(\$event)'> </ej-lineargauge>

DrawMarkerPointers(sender){}| Not Applicable |

| Ranges Event | **Event:** *drawRange*

 <ej-lineargauge (drawRange)= 'DrawRange(\$event)'> </ej-lineargauge>

DrawRange(sender){}| Not Applicable |

| Gauge Initialized Event | **Event:** *init*

 <ej-lineargauge (load)= 'load(\$event)'> </ej-lineargauge>

DrawRange(sender){}| Not Applicable |

ListBox

Getting started with Angular List box component

This section briefly explains how to create a simple **ListBox** component and configure its available functionalities in Angular.

Dependencies

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

```
`javascript
```

```
|-- @syncfusion/ej2-angular-dropdowns
```

```
|-- @syncfusion/ej2-base
```

```
|-- @syncfusion/ej2-data
```

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

Setup Angular environment

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

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

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
ng new my-app
cd my-app
\
```

Installing Syncfusion ListBox 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-dropdowns](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-dropdowns --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-dropdowns@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-dropdowns@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-dropdowns:"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 ListBox module

Import ListBox module into Angular application(`app.module.ts`) from the package

`@syncfusion/ej2-angular-dropdowns`.

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

Adding Syncfusion ListBox component

Modify the template in `app.component.ts` file to render the Button module.

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

```
@Component({
  selector: 'app-root',
  // specifies the template string for the ListBox component
  template: <ejs-listbox></ejs-listbox>,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
`
```

Adding CSS reference

Add Button component's styles as given below in `style.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
`
```

Binding data source

After initialization, populate the ListBox with data using the `dataSource` property.

Here, an array of object is passed to the ListBox component.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the ListBox component
  template: <ejs-listbox [dataSource]='data'></ejs-listbox>
})
export class AppComponent {
  constructor() {
  }
  // defined the array of object
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
    { text: 'SSC Ultimate Aero', id: 'list-04' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
  ]
}
```

```
{ text: 'McLaren F1', id: 'list-06' },
{ text: 'Aston Martin One- 77', id: 'list-07' },
{ text: 'Jaguar XJ220', id: 'list-08' },
{ text: 'McLaren P1', id: 'list-09' },
{ text: 'Ferrari LaFerrari', id: 'list-10' },
];
}
`
```

Run the application

After completing the configuration required to render a basic ListBox, run the following command to display the output in your default browser.

`

ng serve

`

The following example illustrates the output in your browser.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { ListBoxComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the ListBox component with
  dataSource
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]='data'></ejs-listbox></div>`
})
export class AppComponent {
  // defined the array of object
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
    { text: 'SSC Ultimate Aero', id: 'list-04' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
    { text: 'McLaren F1', id: 'list-06' },
    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
```

```
{ text: 'McLaren P1', id: 'list-09' },
{ text: 'Ferrari LaFerrari', id: 'list-10' }
];
}
```

MAIN.TS

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

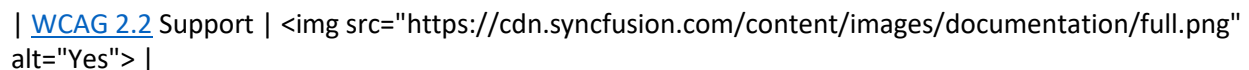
Accessibility in Angular List box component

The List box component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the List box component is outlined below.

| Accessibility Criteria | Compatibility |

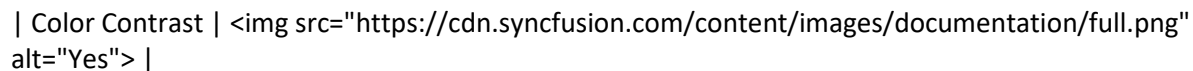
| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes"> |

| [Section 508](#) Support |  alt="Yes"> |

| Screen Reader Support |  alt="Yes"> |

| Right-To-Left Support |  alt="Yes"> |

| Color Contrast |  alt="Yes"> |

| Mobile Device Support |  alt="Yes"> |

| Keyboard Navigation Support |  alt="Yes"> |

| [Accessibility Checker](#) Validation |  alt="Yes"> |

| [Axe-core](#) Accessibility Validation |  alt="Yes"> |

<style>

.post .post-content img {

display: inline-block;

margin: 0.5em 0;

}

</style>

<div> - 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 List box component followed the [WAI-ARIA](https://www.w3.org/WAI/ARIA/apg/patterns/List box/) patterns to meet the accessibility. The following ARIA attributes are used in the List box component:

| Attributes | Purpose |

| --- | --- |

| role | Indicates the List box component wrapper element as List box, the UL element as presentation, and its list item as option. |

| aria-label | Provides an accessible name for the List box component. |

| aria-multiselectable | Applied to the element with the List box role, tells assistive technologies that the list supports multiple selection. The default value is true. |

| aria-selected | Applied to elements with role option that are visually styled as selected to inform assistive technologies that the options are selected. |

Keyboard interaction

The List box component followed the [keyboard interaction](https://www.w3.org/WAI/ARIA/apg/patterns/List box/#keyboardinteraction) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the List box component.

| Press | To do this |

| --- | --- |

| Up arrow | Moves focus to the previous option. |

| Down arrow | Moves focus to the next option. |

| Home | Moves focus to first option. |

| End | Moves focus to last option. |

| Space | Changes the selection state of the focused option. |

| Ctrl + A | Selects all options in the list. |

| Ctrl + Shift + Home | Selects the focused option and all options up to the first option. |

| Ctrl + Shift + End | Selects the focused option and all options down to the last option. |

| Ctrl + (Up or Down) | Press Ctrl key with up / down arrow or mouse to select multiple items. |

Ensuring accessibility

The List box component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the List box component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the List box component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Data binding in Angular List box component

The ListBox loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of `array` or `DataManager`.

Fields	Type	Description
<code>text</code>	<code>string</code>	Specifies the display text of each list item.
<code>value</code>	<code>string</code>	Specifies the hidden data value mapped to each list item that should contain a unique value.
<code>groupBy</code>	<code>string</code>	Specifies the category under which the list item has to be grouped.
<code>iconCss</code>	<code>string</code>	Specifies the iconCss class that needs to be mapped.
<code>htmlAttributes</code>	<code>string</code>	Allows additional attributes to configure the elements in various ways to meet the criteria.

When binding complex data to the ListBox, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Local Data

Local data can be represented by the following ways as described below.

Array of string

The ListBox has support to load array of primitive data such as strings or numbers. Here, both value and text field acts as same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
```



```
template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data"></ejs-listbox></div>`
))
export class AppComponent {
public data: string[] = ['Badminton', 'Cricket', 'Football', 'Golf',
'Tennis', 'Basket Ball', 'Base Ball', 'Hockey', 'Volley Ball'];
}
```

MAIN.TS

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

Array of object

The ListBox can generate its list items through an array of object data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **id** and **sports** column from complex data have been mapped to the **value** field and **text** field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent ,ListBoxAllModule} from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
imports: [
FormsModule, ReactiveFormsModule, ListBoxAllModule
],
standalone: true,
selector: 'app-container',
template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"></ejs-
listbox></div>`
})
export class AppComponent {
public data: { [key: string]: Object }[] = [
{ id: 'game1', sports: 'Badminton' },
{ id: 'game2', sports: 'Cricket'},
{ id: 'game3', sports: 'Football'},
{ id: 'game4', sports: 'Golf'},
{ id: 'game5', sports: 'Tennis'},
{ id: 'game6', sports: 'Basket Ball'},
{ id: 'game7', sports: 'Base Ball'},
{ id: 'game8', sports: 'Hockey'},
{ id: 'game9', sports: 'Volley Ball'}
];
public setfield = {
text: "sports",
```

```

    value: "id"
  }
}

```

MAIN.TS

```

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

```

Array of complex object

The ListBox can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **Sports.Name** column from complex data have been mapped to the **text** field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"></ejs-
listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { Id: 'game1', Sports: { Name: 'Badminton' } },
    { Id: 'game2', Sports: { Name: 'Cricket' } },
    { Id: 'game3', Sports: { Name: 'Football' } },
    { Id: 'game4', Sports: { Name: 'Golf' } },
    { Id: 'game5', Sports: { Name: 'Tennis' } },
    { Id: 'game6', Sports: { Name: 'Basket Ball' } },
    { Id: 'game7', Sports: { Name: 'Base Ball' } },
    { Id: 'game8', Sports: { Name: 'Hockey' } },
    { Id: 'game9', Sports: { Name: 'Volley Ball' } }
  ];
  public setfield = { text: "Sports.Name" }
}

```

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

The ListBox supports retrieval of data from remote data services with the help of [DataManager](#) component. The [Query](#) property is used to fetch data from the database and bind it to the ListBox.

The following sample displays the first 10 products from **Products** table of the **Northwind** Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"
[query]="query"></ejs-listbox></div>`
})
export class AppComponent {
  public data:DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
  })
  public query = new
Query().from('Products').select('ProductID,ProductName').take(10);
  public setfield = { text: "ProductName" }
}
```

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 in Angular List box component

The ListBox has support to drag an item or a group of selected items and drop it within the same list box or into another list box.

The elements can be customized on drag and drop by using the following events,

| Events | Description |

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

| [dragStart](#) | Triggers when the selected element is being dragged. |

| [drag](#) | Triggers when the selected element is being dragged. |

| [drop](#) | Triggers when the selected element is being dropped. |

Single listbox

To drag and drop an item or group of item within the list box can be achieved by setting [allowDragAndDrop](#) property as `true`.

The following sample illustrates how to drag and drop an item within the same list box by enabling `allowDragAndDrop` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component ({
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [allowDragAndDrop]="true"
[fields]="setfield"><ejs-listbox></div>`
})
export class AppComponent {
  public data: {[key: string]: Object}[] = [
    { "Name": "Australia", "Code": "AU" },
    { "Name": "Bermuda", "Code": "BM" },
    { "Name": "Canada", "Code": "CA" },
    { "Name": "Cameroon", "Code": "CM" },
    { "Name": "Denmark", "Code": "DK" },
    { "Name": "France", "Code": "FR" },
    { "Name": "Finland", "Code": "FI" },
    { "Name": "Germany", "Code": "DE" },
    { "Name": "Hong Kong", "Code": "HK" }
  ];
  public setfield = { text : "Name" }
}
```

MAIN.TS

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

Multiple listbox

To drag and drop an item or group of item between two list boxes can be achieved by setting `allowDragAndDrop` property as `true` and [scope](#) property should be set to both the list boxes.

In the following sample, the `allowDragAndDrop` property is set as `true` and `scope` is set as `combined-list` to enable drop and drop in both list boxes.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <div class="drag-drop-wrapper">
      <div class="listbox-control1">
        <h4>Group A</h4>
        <ejs-listbox [dataSource]="groupA"
[allowDragAndDrop]="true" [fields]="setfield" scope="combined-list"
height="290px"></ejs-listbox>
      </div>
      <div class="listbox-control2">
        <h4>Group B</h4>
        <ejs-listbox [dataSource]="groupB"
[allowDragAndDrop]="true" [fields]="setfield" scope="combined-list"
height="290px"></ejs-listbox>
      </div></div>
    </div>`,
})
export class AppComponent {
  public groupA: { [key: string]: Object }[] = [
    { "Name": "Australia", "Code": "AU" },
    { "Name": "Bermuda", "Code": "BM" },
    { "Name": "Canada", "Code": "CA" },
    { "Name": "Cameroon", "Code": "CM" },
    { "Name": "Denmark", "Code": "DK" },
    { "Name": "France", "Code": "FR" },
    { "Name": "Finland", "Code": "FI" },
    { "Name": "Germany", "Code": "DE" },
    { "Name": "Hong Kong", "Code": "HK" }
  ];
  public groupB: { [key: string]: Object }[] = [
    { "Name": "India", "Code": "IN" },
    { "Name": "Italy", "Code": "IT" },
    { "Name": "Japan", "Code": "JP" },
    { "Name": "Mexico", "Code": "MX" },
    { "Name": "Norway", "Code": "NO" },
    { "Name": "Poland", "Code": "PL" },
    { "Name": "Switzerland", "Code": "CH" },
    { "Name": "United Kingdom", "Code": "GB" },
    { "Name": "United States", "Code": "US" }
  ];
}
```

```
];
public setfield = { text: "Name" }
}
```

MAIN.TS

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

Dual list box in Angular List box component

The dual list box allows the user to move items between two list boxes by clicking the toolbar buttons. Dual list box can be created by listing items in the [toolbarSettings](#) along with the `scope` property.

The following operations can be performed in dual list box,

Options	Description
----- -----	
moveUp	Move the selected item in the upward direction within the list box.
moveDown	Move the selected item in the downward direction within the list box.
moveTo	Move the selected item to the another list box.
moveFrom	Move the selected item from one list box to the another list box.
moveAllTo	Move all the items to the another list box.
moveAllFrom	Move all the items from one list box to the another list box.

The following example illustrates how to move items from `Group A` to `Group B` list box.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <div class="dual-list-wrapper">
      <div class="dual-list-groupa">
        <h4>Group A</h4>
        <ejs-listbox [dataSource]="groupA" [fields]="setfield"
height="330px" [toolbarSettings]="toolbar" scope="#listbox"></ejs-listbox>
      </div>
      <div class="dual-list-groupb">
```

```

        <h4>Group B</h4>
        <ejs-listbox [dataSource]="groupB" [fields]="setfield"
height="330px" id="listbox"></ejs-listbox>
    </div></div>
</div>`,
    })
    export class AppComponent {
        public groupA: { [key: string]: Object }[] = [
            { "Name": "Australia", "Code": "AU" },
            { "Name": "Bermuda", "Code": "BM" },
            { "Name": "Canada", "Code": "CA" },
            { "Name": "Cameroon", "Code": "CM" },
            { "Name": "Denmark", "Code": "DK" },
            { "Name": "France", "Code": "FR" },
            { "Name": "Finland", "Code": "FI" },
            { "Name": "Germany", "Code": "DE" },
            { "Name": "Hong Kong", "Code": "HK" }
        ];
        public groupB: { [key: string]: Object }[] = [
            { "Name": "India", "Code": "IN" },
            { "Name": "Italy", "Code": "IT" },
            { "Name": "Japan", "Code": "JP" },
            { "Name": "Mexico", "Code": "MX" },
            { "Name": "Norway", "Code": "NO" },
            { "Name": "Poland", "Code": "PL" },
            { "Name": "Switzerland", "Code": "CH" },
            { "Name": "United Kingdom", "Code": "GB" },
            { "Name": "United States", "Code": "US" }
        ];
        public setfield = { text: "Name" }
        public toolbar = { items: ['moveUp', 'moveDown', 'moveTo', 'moveFrom',
            'moveAllTo', 'moveAllFrom'] }
    }
}

```

MAIN.TS

```

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

```

Icons and templates in Angular List box component**Icons**

To place the icon on a list box, set the [iconCss](#) property to `e-icons` with the required icon CSS. By default, the icon is positioned to the left side of the list.

In the following sample, icon classes are mapped with `iconCss` field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'

```

```
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"></ejs-
listbox></div>`
})
export class AppComponent{
  public data: { [key: string]: Object }[] = [
    { text: 'Account Settings', iconCss: 'e-list-icons e-list-user-settings' },
    { text: 'Address Book', iconCss: 'e-list-icons e-list-address-book' },
    { text: 'Delete', iconCss: 'e-list-icons e-list-delete' },
    { text: 'Forward', iconCss: 'e-list-icons e-list-forward' },
    { text: 'Reply', iconCss: 'e-list-icons e-list-reply' },
    { text: 'Reply All', iconCss: 'e-list-icons e-list-reply-all' },
    { text: 'Save All Attachments', iconCss: 'e-list-icons e-list-save-all-
attachments' },
    { text: 'Save As', iconCss: 'e-list-icons e-list-icon-save-as' },
    { text: 'Touch/Mouse Mode', iconCss: 'e-list-icons e-list-touch' },
    { text: 'Undo', iconCss: 'e-list-icons e-list-undo' }
  ];
  public setfield = { iconCss: 'iconCss' }
}
```

MAIN.TS

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

Templates

ListBox items can be customized according to the requirement using [itemTemplate](#) property.

In the following sample, the items in the cart are displayed using list box template,

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
```



```

standalone: true,
  selector: 'app-container',
  templateUrl: 'template.html',
})
export class AppComponent {
  public data?: { [key: string]: Object }[];
  ngOnInit():void{
    this.data = [
      { text: 'JavaScript', pic: 'javascript', description: 'It is a lightweight interpreted or JIT-compiled programming language.' },
      { text: 'TypeScript', pic: 'typescript', description: 'It is a typed superset of JavaScript that compiles to plain JavaScript.' },
      { text: 'Angular', pic: 'angular', description: 'It is a TypeScript-based open-source web application framework.' },
      { text: 'React', pic: 'react', description: 'A JavaScript library for building user interfaces. It can also render on the server using Node.' },
      { text: 'Vue', pic: 'vue', description: 'A progressive framework for building user interfaces. it is incrementally adoptable.' }
    ];
  }
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div class="e-section-control">
<ejs-listbox [dataSource]="data"
itemTemplate= '<div class="list-wrapper">
    <span class="{pic} e-avatar e-avatar-xlarge e-avatar-circle">
        </span><span class="text">{text}</span>
        <span class="description">{description}</span>
    </div>'>
</ejs-listbox>
</div>

```

Selection in Angular List box component

The ListBox provides support to select an item or a group of item by mouse or keyboard action. There are two selection modes available in list box,

- Single - To select single item in the list box.
- Multiple - To select multiple items in the list box.

On selection of each list box item, [change](#) event is triggered.

Single selection

To enable single selection in the list box, `mode` should be set as `Single` in `selectionSettings` property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data"
    [selectionSettings]="selection"></ejs-listbox></div>`
})
export class AppComponent{
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom' },
    { text: 'Bugatti Chiron' },
    { text: 'Bugatti Veyron Super Sport' },
    { text: 'SSC Ultimate Aero' },
    { text: 'Koenigsegg CCR' },
    { text: 'McLaren F1' },
    { text: 'Aston Martin One- 77' },
    { text: 'Jaguar XJ220' },
    { text: 'McLaren P1' },
    { text: 'Ferrari LaFerrari' }
  ];
  public selection = { mode: "Single" }
}
```

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

To enable multiple selection in the list box, `mode` should be set as `Multiple` in `selectionSettings` property.

To select multiple items, use the SHIFT, CTRL, and arrow keys to make selections.

By default, the selection mode is set as `Multiple`.

APP.COMPONENT.TS

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

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data"
[selectionSettings]="selection"></ejs-listbox></div>`
})
export class AppComponent{
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom' },
    { text: 'Bugatti Chiron' },
    { text: 'Bugatti Veyron Super Sport' },
    { text: 'SSC Ultimate Aero' },
    { text: 'Koenigsegg CCR' },
    { text: 'McLaren F1' },
    { text: 'Aston Martin One- 77' },
    { text: 'Jaguar XJ220' },
    { text: 'McLaren P1' },
    { text: 'Ferrari LaFerrari' }
  ];
  public selection = {
    mode: "Multiple"
  }
}
```

MAIN.TS

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

Checkbox selection

The ListBox supports checkbox in default and grouped list box which is used to select multiple items. CheckBox selection can be enabled by injecting `CheckBoxSelection` module and also [showCheckBox](#) property should be set as `true`.

Select All

To select all the items in the list box, [showSelectAll](#) should be set as `true`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
```

```

import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { ListBoxComponent, CheckBoxSelection } from '@syncfusion/ej2-
angular-dropdowns';
ListBoxComponent.Inject(CheckBoxSelection);
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data"
    [selectionSettings]="selection"></ejs-listbox></div>`
})
export class AppComponent{
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom' },
    { text: 'Bugatti Chiron' },
    { text: 'Bugatti Veyron Super Sport' },
    { text: 'SSC Ultimate Aero' },
    { text: 'Koenigsegg CCR' },
    { text: 'McLaren F1' },
    { text: 'Aston Martin One- 77' },
    { text: 'Jaguar XJ220' },
    { text: 'McLaren P1' },
    { text: 'Ferrari LaFerrari' }
  ];
  public selection = {
    showCheckbox: true,
    showSelectAll: true
  }
}

```

MAIN.TS

```

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

```

To select all the items in the list box, [selectAll](#) method can also be used.

Sorting and grouping in Angular List box component

Sorting

The ListBox supports sorting of available items in the alphabetical order that can be either ascending or descending. This can be achieved using [sortOrder](#) property. Sort order can be **None**, **Ascending** or **Descending**.

In the following example, the **SortOrder** is set as **Descending**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent ,ListBoxAllModule} from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"
sortOrder="Descending"></ejs-listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { "Name": "Australia", "Code": "AU" },
    { "Name": "Bermuda", "Code": "BM" },
    { "Name": "Canada", "Code": "CA" },
    { "Name": "Cameroon", "Code": "CM" },
    { "Name": "Denmark", "Code": "DK" },
    { "Name": "France", "Code": "FR" },
    { "Name": "Finland", "Code": "FI" },
    { "Name": "Germany", "Code": "DE" },
    { "Name": "Hong Kong", "Code": "HK" }
  ];
  public setfield = {
    text: "Name"
  }
}

```

MAIN.TS

```

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

```

Grouping

The ListBox supports to wrap the nested element into a group based on its category. The category of each list item can be mapped with [groupBy](#) field in the data table.

In the following example, vegetables are grouped based on its category.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent , ListBoxAllModule} from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';

```

```
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]="data" [fields]="setfield"></ejs-
listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { "Vegetable": "Cabbage", "Category": "Leafy and Salad", "Id": "item1"
  },
    { "Vegetable": "Spinach", "Category": "Leafy and Salad", "Id": "item2"
  },
    { "Vegetable": "Wheat grass", "Category": "Leafy and Salad", "Id":
"item3" },
    { "Vegetable": "Yarrow", "Category": "Leafy and Salad", "Id": "item4" },
    { "Vegetable": "Pumpkins", "Category": "Leafy and Salad", "Id": "item5"
  },
    { "Vegetable": "Chickpea", "Category": "Beans", "Id": "item6" },
    { "Vegetable": "Green bean", "Category": "Beans", "Id": "item7" },
    { "Vegetable": "Horse gram", "Category": "Beans", "Id": "item8" },
    { "Vegetable": "Garlic", "Category": "Bulb and Stem", "Id": "item9" },
    { "Vegetable": "Nopal", "Category": "Bulb and Stem", "Id": "item10" },
    { "Vegetable": "Onion", "Category": "Bulb and Stem", "Id": "item11" }
  ];
  public setfield = {
    groupBy: "Category",
    text: "Vegetable",
    value: "Id"
  }
}
```

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 List box component

To modify the ListBox appearance, you need to override the default CSS of ListBox component. Please find the list of CSS classes and its corresponding section in ListBox component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

| CSS Class | Purpose of Class |

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

|.e-listbox-wrapper|To customize the listbox wrapper |

|.e-list-parent .e-list-item|To customize the listbox list items |

|.e-list-parent .e-list-item:hover|To customize the listbox list items on hover |

|.e-list-parent .e-list-item.e-selected|To customize the listbox selected list item |

|.e-listboxtool-wrapper .e-listbox-tool|To customize the listbox toolbar |

|.e-listboxtool-wrapper .e-listbox-tool .e-btn|To customize the listbox toolbar button |

|.e-listboxtool-wrapper .e-listbox-tool .e-btn .e-btn-icon.e-icons::before|To customize the listbox toolbar icon |

Horizontal ListBox

You can use [cssClass](#) property to display the Listbox horizontally.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { ListBoxComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the ListBox component with
  // dataSource
  template: `<div class="e-section-control">
    <ejs-listbox [dataSource]='data' cssClass='e-horizontal-listbox'></ejs-listbox></div>`
})
export class AppComponent {
  // defined the array of object
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
    { text: 'SSC Ultimate Aero', id: 'list-04' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
    { text: 'McLaren F1', id: 'list-06' },
    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
    { text: 'McLaren P1', id: 'list-09' },
    { text: 'Ferrari LaFerrari', id: 'list-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

Add items in Angular List box component

To add an item or multiple items, [addItem](#) method can be used. In the following example, the **Bugatti Veyron Super Sport** and **SSC Ultimate Aero** items will be added while clicking **Add Items** button.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { getInstance } from '@syncfusion/ej2-base';
import { ListBox } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <button (click)="btnclick()" class="e-btn">ADD
ITEMS</button>
    <ejs-listbox id="listbox" [dataSource]="data"></ejs-
listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
    { text: 'McLaren F1', id: 'list-06' },
    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
    { text: 'McLaren P1', id: 'list-09' },
    { text: 'Ferrari LaFerrari', id: 'list-10' }
  ];
  btnclick(){
    let listboxobj:ListBox = getInstance((document as
any).getElementById("listbox"), ListBox) as ListBox;
    let addItem: { [key: string]: Object }[] = [
      { text: 'Bugatti Veyron Super Sport', id: 'list-03' }, { text: 'SSC
Ultimate Aero', id: 'list-04' }
    ];
    if (!listboxobj.getDataByValue('Bugatti Veyron Super Sport')) {
      listboxobj.addItem(addItem);
    }
  }
}
```

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 items in Angular List box component

To enable or disable items in the list box, [enableItems](#) method can be used. In the following example, the Bugatti Veyron Super Sport and SSC Ultimate Aero items are disabled by default and by clicking Enable Items buttons, the disabled items will be enabled.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { getInstance } from '@syncfusion/ej2-base';
import { ListBox } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <button id="enableitem" (click)="btnclick()" class="e-btn">ENABLE ITEMS</button>
    <ejs-listbox id="listbox" [dataSource]="data"
    (created)="created()"></ejs-listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
    { text: 'SSC Ultimate Aero', id: 'list-04' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
    { text: 'McLaren F1', id: 'list-06' },
    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
    { text: 'McLaren P1', id: 'list-09' },
    { text: 'Ferrari LaFerrari', id: 'list-10' }
  ];
  created():void{
    let listboxobj:ListBox = getInstance((document as any).getElementById("listbox"), ListBox) as ListBox;
    listboxobj.enableItems(['Bugatti Veyron Super Sport', 'SSC Ultimate Aero'], false);
  }
  btnclick():void{
    let listboxobj:ListBox = getInstance((document as any).getElementById("listbox"), ListBox) as ListBox;
    listboxobj.enableItems(['Bugatti Veyron Super Sport', 'SSC Ultimate Aero'], 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 scroller in Angular List box component

The ListBox supports scrolling and it can be achieved by restricting the height of the list box using [height](#) property.

In the following sample, **height** of the list box is restricted to **290px**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox id="listbox" [dataSource]="data"
height="290px"></ejs-listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { text: 'Account Settings', id: 'list-01' },
    { text: 'Address Book', id: 'list-02' },
    { text: 'Delete', id: 'list-03' },
    { text: 'Forward', id: 'list-04' },
    { text: 'Reply', id: 'list-05' },
    { text: 'Reply All', id: 'list-06' },
    { text: 'Save All Attachments', id: 'list-07' },
    { text: 'Save As', id: 'list-08' },
    { text: 'Touch/Mouse Mode', id: 'list-09' },
    { text: 'Undo', id: 'list-10' }
  ];
}
```

MAIN.TS

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

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

Form submit in Angular List box component

In the following code snippet, the value that is in selected state will be sent on form submit.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { ListBoxComponent, ListBoxAllModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <form><ejs-listbox id="listbox" [dataSource]="data"
height="290px"></ejs-listbox>
    <button class="e-btn">Submit</button></form></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom' },
    { text: 'Bugatti Chiron' },
    { text: 'Bugatti Veyron Super Sport' },
    { text: 'SSC Ultimate Aero' },
    { text: 'Koenigsegg CCR' },
    { text: 'McLaren F1' },
    { text: 'Aston Martin One- 77' },
    { text: 'Jaguar XJ220' },
    { text: 'McLaren P1' },
    { text: 'Ferrari LaFerrari' },
  ];
}
```

MAIN.TS

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

Select items in Angular List box component

In the following example, Bugatti Chiron is selected using [selectItems](#) method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
```

```

import { ListBoxComponent,ListBoxAllModule } from '@syncfusion/ej2-angular-
dropdowns'
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { getInstance } from '@syncfusion/ej2-base';
import { ListBox } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule,ListBoxAllModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<div class="e-section-control">
    <ejs-listbox id="listbox" [dataSource]="data"
    (created)="created()"></ejs-listbox></div>`
})
export class AppComponent {
  public data: { [key: string]: Object }[] = [
    { text: 'Hennessey Venom', id: 'list-01' },
    { text: 'Bugatti Chiron', id: 'list-02' },
    { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
    { text: 'SSC Ultimate Aero', id: 'list-04' },
    { text: 'Koenigsegg CCR', id: 'list-05' },
    { text: 'McLaren F1', id: 'list-06' },
    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
    { text: 'McLaren P1', id: 'list-09' },
    { text: 'Ferrari LaFerrari', id: 'list-10' }
  ];
  created():void{
    let listboxobj:ListBox = getInstance((document as
    any).getElementById("listbox"), ListBox) as ListBox;
    listboxobj.selectItems(['Bugatti Chiron']);
  }
}

```

MAIN.TS

```

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

```

ListView

Getting started with Angular Listview component

The ListView component is available in `@syncfusion/ej2-angular-lists` package. Utilize this package to render the ListView Component.

Setup Angular Environment

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

To install Angular CLI use the following command.

```
`bash
```

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

```
,
```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```
`bash
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion Listview package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-lists --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-lists@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-lists@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-lists:"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 ListView Module

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

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import Listview component Module
import { ListViewModule } from '@syncfusion/ej2-angular-lists';
import { AppComponent } from './app.component';
@NgModule({
  //declaration of listview module into NgModule
  imports: [ BrowserModule, ListViewModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding CSS Reference

- Add ListView component's styles as given below in `styles.css`.

```
`css
@import "../node_modules/@syncfusion/ej2-base/styles/material.css";
@import "../node_modules/@syncfusion/ej2-angular-lists/styles/material.css";
```

- If you are using `CheckList` behaviour in ListView, we need to add `Button` component's styles as given below in `styles.css` file

```
`css
@import "../node_modules/@syncfusion/ej2-angular-buttons/styles/material.css";
```

We can also use [CRG](#) to generate combined component styles.

Add Listview component

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

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

```
`typescript
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the Listview component
  template: <ejs-listview id='sample-list' [dataSource]='data'></ejs-listview>
})
export class AppComponent {
  public data: Object = [
    { text: 'Artwork', id: '01' },
    { text: 'Abstract', id: '02' },
    { text: 'Modern Painting', id: '03' },
    { text: 'Ceramics', id: '04' },
    { text: 'Animation Art', id: '05' },
    { text: 'Oil Painting', id: '06' }];
}
```

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 { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
```

```

    selector: 'my-app',
    template: `<ejs-listview id='sample-list' [dataSource]='data'></ejs-
listview>`
  })
  export class AppComponent {
    public data: Object = [
      { text: 'Artwork', id: '01' },
      { text: 'Abstract', id: '02' },
      { text: 'Modern Painting', id: '03' },
      { text: 'Ceramics', id: '04' },
      { text: 'Animation Art', id: '05' },
      { text: 'Oil Painting', id: '06' }
    ];
  }

```

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

[Data Binding Types](#)

[ListView customization](#)

[Virtualization](#)

Schematics in Angular Listview component

Angular schematics is a workflow tool that allows you generate component, modules, and resolve dependency issues. The main goal of the schematics is ease of use and development in angular environment.

EJ2 ListView supports Angular schematic's module injection, component generation, automation dependency installation, styles imports, etc.

Getting started

Note: Angular schematics supports only from the Angular CLI v6. So, check your version by running `ng --version`. If it is below version 6, update your CLI by running the following command: `npm install -g @angular/cli`.

In order to work with Angular schematics, create an Angular CLI application. Run the following command to create a CLI application.

`

`ng new angular-application`

`

After running the above command and all the dependency modules installed, we can generate EJ2 ListView component using schematics.

Dependency and Module injection using Schematics

Using schematics, we can perform dependency and module injection of the EJ2 lists package `@syncfusion/ej2-angular-lists` automatically. Run the following command in the root of the application.

```
ng add @syncfusion/ej2-angular-lists
```

The above command will do the following,

- Installs the `@syncfusion/ej2-angular-lists` package, and add an entry in `package.json` file.
- Imports the `ListViewModule` module in the `app.module.ts`, and add an entry in the `import` property of the `@NgModule` decorator.

Component generation using Schematics

Angular Schematics can be used to generate component, module file, etc. In the same way, ListView components can also be generated.

By using the Schematics to generate EJ2 ListView, the time for configuring components is significantly reduced and it is made ready for development immediately. To generate EJ2 ListView component with specific features, refer to the following table.

The general syntax for the `ng generate` command is

```
ng generate @syncfusion/<component-package-name>:<componentName-featureName> --name=<your-desired-name>
```

Feature Name	Schematics command
Default List	<code>ng generate @syncfusion/ej2-angular-lists:listview-default --name=default-list</code>
Check List	<code>ng generate @syncfusion/ej2-angular-lists:listview-checklist --name=check-list</code>
Nested List	<code>ng generate @syncfusion/ej2-angular-lists:listview-nestedlist --name=nested-list</code>
Remote List	<code>ng generate @syncfusion/ej2-angular-lists:listview-remotelist --name=remote-list</code>
Template List	<code>ng generate @syncfusion/ej2-angular-lists:listview-template --name=template-list</code>
Virtualization	<code>ng generate @syncfusion/ej2-angular-lists:listview-virtualization --name=virtualization-list</code>

The above commands will do the following,

- Generate the ListView with specific features mentioned in the `src/app` with folder name of the `name` property passed.
- Import the generated component into the `app.module.ts` file, and add an entry in the `declarations` array in the `@NgModule` decorator.

Note: It is not required to run the above commands only after the `ng add @syncfusion/ej2-angular-lists` but it is required to have `@syncfusion/ej2-angular-lists` installed.

Data binding in Angular Listview component

ListView provides the option to load the data either from local data sources or remote data services.

This can be done through `dataSource` property which supports the data type of array or through `DataManager`.

ListView supports different kind of data services such as OData, OData V4, Web API and data formats like XML, JSON, JSONP with the help of `DataManager` Adaptors.

Fields	Type	Description
id	string	Specifies ID attribute of list item, mapped in <code>dataSource</code> .
text	string	Specifies list item display text field.
isChecked	boolean	Specifies checked status of list item.
isVisible	boolean	Specifies visibility state of list item.
enabled	boolean	Specifies enabled state of list item.
iconCss	string	Specifies the icon class of each list item which will be add before to inner text.
child	string	Specifies child <code>dataSource</code> fields.
tooltip	string	Specifies tooltip title text field.
groupBy	string	Specifies category of each list item.
sortBy	string	Specifies sorting field, which is used to sort the listview data.
htmlAttributes	string	Specifies list item html attributes field.

When complex data bind to ListView, you should map the fields properly. Otherwise, the ListView properties remain as undefined or null.

Bind to local data

Local data represents in two ways, which are described below.

- Array of simple data
- Array of JSON data.

Array of simple data

ListView supports to load the array of primitive data like string and numbers. Here, both value and text field act as same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data'></ejs-listview>`,
})
export class AppComponent {
  //define the array of string
  public data: string[] = ["Artwork", "Abstract", "Modern Painting", "Ceramics", "Animation Art", "Oil Painting"];
}
```

MAIN.TS

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

Array of JSON data

ListView can generate its list items through an array of complex data. To get it work properly, you should map the appropriate columns to field property.

In below example, role column has mapped with text field.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data' [fields]='fields' [showHeader]='true' [headerTitle]='headertitle'></ejs-listview>`,
})
export class AppComponent {
  public data: Object = [
    {
      'Name': 'Display',
      'id': 'list-01'
    },
  ],
```

```

    {
      'Name': 'Notification',
      'id': 'list-02'
    },
    {
      'Name': 'Sound',
      'id': 'list-03'
    },
    {
      'Name': 'Apps',
      'id': 'list-04'
    },
    {
      'Name': 'Storage',
      'id': 'list-05'
    },
    {
      'Name': 'Battery',
      'id': 'list-06'
    }
  ];
  public fields: Object = { text: 'Name', tooltip: 'Name', id:'id' };
  public headertitle = 'Device settings';
}

```

MAIN.TS

```

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

```

Bind to remote data

ListView supports to retrieve the data from remote data services with the help of DataManager component and Query property allows to fetch data and return it to ListView from the database.

In the below sample, displayed first 6 Products from Product table of NorthWind data service.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
//import DataManager related classes
import { DataManager, Query, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[query]='query' [fields]='fields' [showHeader]='true'
[headerTitle]='headertitle'></ejs-listview>`,

```

```

    })
    export class AppComponent {
        public data: Object = new DataManager({
            url: '///js.syncfusion.com/ejServices/Wcf/Northwind.svc/',
            crossDomain: true
        });
        public query = new
        Query().from('Products').select('ProductID,ProductName').take(6);
        public fields: Object = { id: 'ProductID', text: 'ProductName' };
        public headertitle = 'Product Name';
    }

```

MAIN.TS

```

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

```

Grouping in Angular Listview component

ListView supports to wrap the nested element into a group based on category.

The category of each list item can be mapped with `groupBy` field in the data table, which also supports single-level navigation.

In below sample, Cars are grouped based on its category using `groupBy` field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' ></ejs-listview>`
})
export class AppComponent {
  public data: Object = [
    {
      'text': 'Audi A4',
      'id': '9bdb',
      'category': 'Audi'
    },
    {
      'text': 'Audi A5',
      'id': '4589',
      'category': 'Audi'
    },
  ],
}

```

```

        'text': 'BMW 501',
        'id': 'f849',
        'category': 'BMW'
    },
    {
        'text': 'BMW 502',
        'id': '7aff',
        'category': 'BMW'
    }
];
public fields: Object = { groupBy: 'category', tooltip: 'text' };
}

```

MAIN.TS

```

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

```

Customization

The grouping header can be customized by using groupTemplate property for both inline and fixed group header.

Check list in Angular Listview component

The ListView supports checkbox in default and group-lists which is used to select multiple items.

The checkbox can be enabled by the `showCheckBox` property.

The Checkbox will be useful in the scenario where we need to select multiple options. For Example,

In Shipping cart we can be able to select or unselect the desired items before checkout and also it will be useful in selecting multiple items that belongs to same category using the group list.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ListViewModule } from '@syncfusion/ej2-angular-lists';
import { Component } from '@angular/core';
@Component({
    imports: [
        ListViewModule
    ],
    standalone: true,
    selector: 'my-app',
    template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' [showCheckBox]='true' headerTitle='To Do List'
showHeader='true'></ejs-listview>`,
})
export class AppComponent {
    public data: Object = [
        {text: 'Do Meditation', id: '1'},
        {text: 'Go Jogging', id: '2'},
        {text: 'Buy Groceries', id: '3'},
    ]
}

```

```

        {text: 'Pay Phone bill', id: '4'},
        {text: 'Play Football with your friends', id: '5'},
    ];
    public fields: Object = { text: 'text', id:'id' };
}

```

MAIN.TS

```

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

```

Checkbox Position

In ListView the checkbox can be positioned into either **Left** or **Right** side of the list-item text.

This can be achieved by **checkboxPosition** property. By default, checkbox will be positioned to **Left** of list-item text.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' [showCheckBox]='true' [checkboxPosition]='position"></ejs-
listview>`,
})
export class AppComponent {
  public data: Object = [
    { 'text': 'Hennessey Venom', 'id': 'list-01' },
    { 'text': 'Bugatti Chiron', 'id': 'list-02' },
    { 'text': 'Bugatti Veyron Super Sport', 'id': 'list-03' },
    { 'text': 'SSC Ultimate Aero', 'id': 'list-04' },
    { 'text': 'Koenigsegg CCR', 'id': 'list-05' },
    { 'text': 'McLaren F1', 'id': 'list-06' },
    { 'text': 'Aston Martin One- 77', 'id': 'list-07' },
    { 'text': 'Jaguar XJ220', 'id': 'list-08' },
    { 'text': 'McLaren P1', 'id': 'list-09' },
    { 'text': 'Ferrari LaFerrari', 'id': 'list-10' }
  ];
  public fields: Object = { text: 'text', id:'id' };
  public 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));
```

Nested list in Angular Listview component

ListView component supports Nested list. For that, we should define child property for the nested list in array of JSON.

The below samples illustrates, how the list is generated with the nested source.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='sample-list' [dataSource]='data'
[fields]='fields' [showHeader]='true' [headerTitle]='headertitle'></ejs-
listview>`,
})
export class AppComponent {
  public data: Object = [
    {
      'text': 'Asia',
      'id': '01',
      'category': 'Continent',
      'child': [{
        'text': 'India',
        'id': '1',
        'category': 'Asia',
        'child': [{
          'text': 'Delhi',
          'id': '1001',
          'category': 'India',
        },
        {
          'text': 'Kashmir',
          'id': '1002',
          'category': 'India',
        },
        {
          'text': 'Goa',
          'id': '1003',
          'category': 'India',
        },
      ]
    },
    {

```



```

        'text': 'China',
        'id': '2',
        'category': 'Asia',
        'child': [{
            'text': 'Zhejiang',
            'id': '2001',
            'category': 'China',
        },
        {
            'text': 'Hunan',
            'id': '2002',
            'category': 'China',
        },
        {
            'text': 'Shandong',
            'id': '2003',
            'category': 'China',
        }
    ]
  }],
},
{
  'text': 'North America',
  'id': '02',
  'category': 'Continent',
  'child': [{
    'text': 'USA',
    'id': '3',
    'category': 'North America',
    'child': [{
      'text': 'California',
      'id': '3001',
      'category': 'USA',
    },
    {
      'text': 'New York',
      'id': '3002',
      'category': 'USA',
    },
    {
      'text': 'Florida',
      'id': '3003',
      'category': 'USA',
    }
    ]
  }],
},
{
  'text': 'Canada',
  'id': '4',
  'category': 'North America',
  'child': [{
    'text': 'Ontario',
    'id': '4001',
    'category': 'Canada',
  },
  {
    'text': 'Alberta',
    'id': '4002',
    'category': 'Canada',
  }

```

```

    },
    {
        'text': 'Manitoba',
        'id': '4003',
        'category': 'Canada',
    }
  ]
},
{
  'text': 'Europe',
  'id': '03',
  'category': 'Continent',
  'child': [{
    'text': 'Germany',
    'id': '5',
    'category': 'Europe',
    'child': [{
      'text': 'Berlin',
      'id': '5001',
      'category': 'Germany',
    },
    {
      'text': 'Bavaria',
      'id': '5002',
      'category': 'Germany',
    },
    {
      'text': 'Hesse',
      'id': '5003',
      'category': 'Germany',
    }
  ]
}, {
  'text': 'France',
  'id': '6',
  'category': 'Europe',
  'child': [{
    'text': 'Paris',
    'id': '6001',
    'category': 'France',
  },
  {
    'text': 'Lyon',
    'id': '6002',
    'category': 'France',
  },
  {
    'text': 'Marseille',
    'id': '6003',
    'category': 'France',
  }
  ]
}
  ]
},
{
  'text': 'Australia',
  'id': '04',
  'category': 'Continent',
  'child': [{

```

```
'text': 'Australia',  
  'id': '7',  
  'category': 'Australia',  
  'child': [{  
    'text': 'Sydney',  
    'id': '7001',  
    'category': 'Australia',  
  },  
  {  
    'text': 'Melbourne',  
    'id': '7002',  
    'category': 'Australia',  
  },  
  {  
    'text': 'Brisbane',  
    'id': '7003',  
    'category': 'Australia',  
  }]  
}, {  
  'text': 'New Zealand',  
  'id': '8',  
  'category': 'Australia',  
  'child': [{  
    'text': 'Milford Sound',  
    'id': '8001',  
    'category': 'New Zealand',  
  },  
  {  
    'text': 'Tongariro National Park',  
    'id': '8002',  
    'category': 'New Zealand',  
  },  
  {  
    'text': 'Fiordland National Park',  
    'id': '8003',  
    'category': 'New Zealand',  
  }]  
}]  
},  
{  
  'text': 'Africa',  
  'id': '05',  
  'category': 'Continent',  
  'child': [{  
    'text': 'Morocco',  
    'id': '9',  
    'category': 'Africa',  
    'child': [{  
      'text': 'Rabat',  
      'id': '9001',  
      'category': 'Morocco',  
    },  
    {  
      'text': 'Toubkal',  
      'id': '9002',  
      'category': 'Morocco',  
    },  
  ],  
  },  
]
```

```

        {
            'text': 'Todgha Gorge',
            'id': '9003',
            'category': 'Morocco',
        }
    ], {
        'text': 'South Africa',
        'id': '10',
        'category': 'Africa',
        'child': [{
            'text': 'Cape Town',
            'id': '10001',
            'category': 'South Africa',
        },
        {
            'text': 'Pretoria',
            'id': '10002',
            'category': 'South Africa',
        },
        {
            'text': 'Bloemfontein',
            'id': '10003',
            'category': 'South Africa',
        }
    ]
    }
    ];
    public fields: {[key: string]: string} = { tooltip: 'text' };
    public headertitle = 'Continent';
}

```

MAIN.TS

```

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

```

Customizing templates in Angular Listview component

The ListView component is designed to customize list items, group title and header title.

Header template

ListView header can be customized with the help of the [headerTemplate](#) property.

To customize header template in your application, set your customized template element inside ng tag directive along with [showHeader](#) property as `true` to display the ListView header.

In the following example, we have rendered Listview with customized header which contains search, add and sort buttons.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ListViewModule } from '@syncfusion/ej2-angular-lists';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';

```

```

import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [

    ListViewModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
<div id = 'flat-list'>
  <!-- ListView element -->
  <ejs-listview id='element' [dataSource]='data' showHeader='true'>
    <ng-template #headerTemplate let-data="">
      <div class="headerContainer"><span
class="fruitHeader">Fruits</span><button ejs-button id="search" iconCss='e-
icons e-search-icon' cssClass='e-small e-round'
isPrimary='true'></button><button ejs-button id="add" iconCss='e-icons e-
add-icon' cssClass='e-small e-round' isPrimary='true'></button><button ejs-
button id="sort" iconCss='e-icons e-sort-icon' cssClass='e-small e-round'
isPrimary='true'></button></div>
    </ng-template>
  </ejs-listview>
</div>
  `,
})
export class AppComponent {
  public data=[
    { text: 'Date', id: '1', imgUrl: './dates.jpg' },
    { text: 'Fig', id: '2', imgUrl: './fig.jpg' },
    { text: 'Apple', id: '3', imgUrl: './apple.png' },
    { text: 'Apricot', id: '4', imgUrl: './apricot.jpg' },
    { text: 'Grape', id: '5', imgUrl: './grape.jpg' },
    { text: 'Strawberry', id: '6', imgUrl: './strawberry.jpg' },
    { text: 'Pineapple', id: '7', imgUrl: './pineapple.jpg' },
    { text: 'Melon', id: '8', imgUrl: './melon.jpg' },
    { text: 'Lemon', id: '9', imgUrl: './lemon.jpg' },
    { text: 'Cherry', id: '10', imgUrl: './cherry.jpg' },
  ];
}

```

MAIN.TS

```

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

```

Template

ListView items can be customized with the help of the [template](#) property.

To customize list items in your application, set your customized template element inside ng tag directive.

We provided the following built-in CSS classes to customize the list-items. Refer to the following table.

CSS class	Description
-----------	-------------

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

| e-list-template, e-list-wrapper | These classes are used to differentiate normal and template rendering, which are mandatory for template rendering. The `e-list-template` class should be added to the root of the ListView element and `e-list-wrapper` class should be added to the template element wrapper.

| e-list-content | This class is used to align list content and it should be added to the content element
`

 <div class="e-list-wrapper">
ListItem
</div>`|

| e-list-avatar | This class is used for avatar customization. It should be added to the template element wrapper. After adding it, we can customize our element with [Avatar](#) classes
`

 <div class="e-list-wrappere-list-avatar">
 MR
ListItem
</div>`|

| e-list-avatar-right | This class is used to align avatar to right side of the list item. It should be added to the template element wrapper. After adding it, we can customize our element with [Avatar](#) classes
`

 <div class="e-list-wrappere-list-avatar-right">
 ListItem
MR
</div>`|

| e-list-badge | This class is used for badge customization .It should be added to the template element wrapper. After adding it, we can customize our element with [Badge](#) classes
`

 <div class="e-list-wrappere-list-badge">
 ListItem
MR
</div>`|

| e-list-multi-line | This class is used for multi-line customization. It should be added to the template element wrapper. After adding it, we can customize List item's header and description
`

<div class="e-list-wrappere-list-multi-line">
 ListItem
</div>`|

| e-list-item-header | This class is used to align a list header and it should be added to the header element along with the multi-line class
`

 <div class="e-list-wrappere-list-multi-line">
 ListItem Header
 ListItem
</div>`|

In the following example, we have customized list items with built-in CSS classes.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
```

```

    template: `
    <div id="sample">
      <ejs-listview id='List' [dataSource]='data' headerTitle='Contacts'
cssClass='e-list-template' [showHeader]='true' sortOrder='Ascending'>
        <ng-template #template let-data="">
          <div class="e-list-wrapper e-list-multi-line e-list-avatar">
            <span class="e-avatar e-avatar-circle"
*ngIf="data.avatar !== ''">{{data.avatar}}</span>
            <span class="{{data.pic}} e-avatar e-avatar-circle"
*ngIf="data.pic !== ''"> </span>
            <span class="e-list-item-header">{{data.text}}</span>
            <span class="e-list-content">{{data.contact}}</span>
          </div>
        </ng-template>
      </ejs-listview>
    </div>
    `
  })
  export class AppComponent {
    // Listview datasource with avatar and image source fields
    public data?: { [key: string]: Object; }[] = [
      {
        text: "Jenifer",
        contact: "(206) 555-985774",
        id: "1",
        avatar: "",
        pic: "pic01"
      },
      { text: "Amenda", contact: "(206) 555-3412", id: "2", avatar: "A", pic: "" },
    ],
    {
      text: "Isabella",
      contact: "(206) 555-8122",
      id: "4",
      avatar: "",
      pic: "pic02"
    },
    {
      text: "William ",
      contact: "(206) 555-9482",
      id: "5",
      avatar: "W",
      pic: ""
    },
    {
      text: "Jacob",
      contact: "(71) 555-4848",
      id: "6",
      avatar: "",
      pic: "pic04"
    },
    { text: "Matthew", contact: "(71) 555-7773", id: "7", avatar: "M", pic: "" },
  ],
  {
    text: "Oliver",
    contact: "(71) 555-5598",
    id: "8",
  }

```

```

    avatar: "",
    pic: "pic03"
  },
  {
    text: "Charlotte",
    contact: "(206) 555-1189",
    id: "9",
    avatar: "C",
    pic: ""
  }
];
}

```

MAIN.TS

```

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

```

Group template

ListView group header can be customized with the help of the [groupTemplate](#) property.

To customize the group template in your application, set your customized template element inside ng tag directive.

In the following example, we have grouped Listview based on the category. The category of each list item should be mapped with [groupBy](#) field of the data. We have also displayed grouped list items count in the group list header.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='List' [dataSource]='dataSource'
cssClass='e-list-template' [fields]='fields'>
  <ng-template #template let-dataSource="">
    <div class="e-list-wrapper e-list-multi-line e-list-avatar">
      <img class="e-avatar e-avatar-circle" src={{dataSource.image}}
style="background:#BCBCBC" />
      <span class="e-list-item-header">{{dataSource.Name}}</span>
      <span class="e-list-content">{{dataSource.contact}}</span>
    </div>
  </ng-template>
  <ng-template #groupTemplate let-dataSource="">
    <div>

```



```

    <span className="category">{{ dataSource.items[0].category
  }}</span>
    <span id="count"> {{ dataSource.items.length }} Item(s)</span>
  </div>
</ng-template>
</ejs-listview>`
  })
  export class AppComponent {
    public dataSource: Object = [
      { Name: 'Nancy', contact: '(206) 555-985774', id: '1', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/1.png', category:
        'Experience' },
      { Name: 'Janet', contact: '(206) 555-3412', id: '2', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/3.png', category:
        'Fresher' },
      { Name: 'Margaret', contact: '(206) 555-8122', id: '4', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/4.png', category:
        'Experience' },
      { Name: 'Andrew ', contact: '(206) 555-9482', id: '5', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/2.png', category:
        'Experience' },
      { Name: 'Steven', contact: '(71) 555-4848', id: '6', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/5.png', category:
        'Fresher' },
      { Name: 'Michael', contact: '(71) 555-7773', id: '7', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/6.png', category:
        'Experience' },
      { Name: 'Robert', contact: '(71) 555-5598', id: '8', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/7.png', category:
        'Fresher' },
      { Name: 'Laura', contact: '(206) 555-1189', id: '9', image:
        'https://ej2.syncfusion.com/demos/src/grid/images/8.png', category:
        'Experience' },
    ];
    public fields: Object = { text: 'Name', groupBy: 'category' };
  }

```

MAIN.TS

```

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

```

Virtualization in Angular Listview component

UI virtualization loads only viewable list items in a view port which will increase ListView performance on loading large number of data.

Module injection

In order to use UI Virtualization, we need to import `VirtualizationService` module in the AppModule and it should be injected to the provider section as follow

`typescript

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { ListViewModule, VirtualizationService } from '@syncfusion/ej2-angular-lists';
/

    • Module

*/
@NgModule({
imports: [
BrowserModule,
ListViewModule
],
declarations: [AppComponent],
bootstrap: [AppComponent],
providers: [VirtualizationService]
})
export class AppModule { }
`
```

Getting started

UI virtualization can be enabled in ListView by setting `enableVirtualization` property to true.

It has two type of scroller

Window scroll - This scroller is used in ListView by default.

Container scroll - This will be used, if the height property of ListView was set.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule, VirtualizationService } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
imports: [

    ListViewModule
],
providers: [VirtualizationService],
standalone: true,
selector: 'my-app',
```

```

    template: `<ejs-listview id='ui-list' [dataSource]='listData'
[enableVirtualization]='true' ></ejs-listview>`
  })
  export class AppComponent {
    public listData: { [key: string]: string | object }[] = [
      { text: 'Nancy', id: '0' },
      { text: 'Andrew', id: '1' },
      { text: 'Janet', id: '2' },
      { text: 'Margaret', id: '3' },
      { text: 'Steven', id: '4' },
      { text: 'Laura', id: '5' },
      { text: 'Robert', id: '6' },
      { text: 'Michael', id: '7' },
      { text: 'Albert', id: '8' },
      { text: 'Nolan', id: '9' }
    ];
    public ngOnInit() {
      for (let i: number = 10; i <= 1010; i++) {
        let index: number = parseInt((Math.random() * 10).toString());
        this.listData.push({ text: (this.listData[index] as any).text,
id: i.toString() });
      }
    }
  }
}

```

MAIN.TS

```

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

```

We can use `ng-template` property to customize list items in UI virtualization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { ListViewModule, VirtualizationService } from '@syncfusion/ej2-angular-lists';
import { Component } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  providers: [VirtualizationService],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='ui-list' [dataSource]='listData'
[enableVirtualization]='true' [height]=500 headerTitle='Contacts'
[showHeader]='true'>
  <ng-template #template let-data="">
    <div class="list-container">
      <div id="icon" *ngIf="data.icon !== ''" class={{data.icon}}>
        <span class="showUI">{{data.icon}}</span>

```

```

        <img class="hideUI" width = '45' height = '45'
src={{data.imageUrl}} />
    </div>
    <div id="icon" *ngIf="data.imageUrl !== ''" class="img">
        <span class="hideUI">{{data.icon}}</span>
        <img class="showUI" width = '45' height = '45'
src={{data.imageUrl}} />
    </div>
    <div class="name">{{data.name}}</div>
    </div>
</ng-template>
</ejs-listview>`
})
export class AppComponent {
    public listData: { [key: string]: string | object }[] = [
        { name: 'Nancy', icon: 'N', imageUrl: '', id: '0' },
        { name: 'Andrew', icon: 'A', imageUrl: '', id: '1' },
        { name: 'Janet', icon: 'J', imageUrl: '', id: '2' },
        { name: 'Margaret', icon: '', imageUrl:
'https://ej2.syncfusion.com/demos/src/grid/images/2.png', id: '3' },
        { name: 'Steven', icon: 'S', imageUrl: '', id: '4' },
        { name: 'Laura', icon: '', imageUrl:
'https://ej2.syncfusion.com/demos/src/grid/images/3.png', id: '5' },
        { name: 'Robert', icon: 'R', imageUrl: '', id: '6' },
        { name: 'Michael', icon: 'M', imageUrl: '', id: '7' },
        { name: 'Albert', icon: '', imageUrl:
'https://ej2.syncfusion.com/demos/src/grid/images/5.png', id: '8' },
        { name: 'Nolan', icon: 'N', imageUrl: '', id: '9' }
    ];
    public ngOnInit() {
        for (let i: number = 10; i <= 1010; i++) {
            let index: number = parseInt((Math.random() * 10).toString());
            this.listData.push({ name: (this.listData[index] as any).name, icon:
(this.listData[index] as any).icon, imageUrl: (this.listData[index] as
any).imageUrl, id: i.toString() });
        }
    }
}

```

MAIN.TS

```

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

```

Scrolling in Angular Listview component

Scrolling is a technique that allows you to load more items as the user scrolls through a list, providing a seamless and dynamic user experience.

Render the ListView with `dataSource`, and bind the [scroll](#) event. Within the scroll event, you can access information such as the scroll direction, event name and the distance from the scrollbar to the top and bottom ends through the `distanceY` parameter.

In the given example, new data is seamlessly added while scrolling. When you scrolls to the bottom and the distance scrolled is less than 100 pixels, it dynamically loads a batch of items into the list as long as there are more items to render.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { ListViewComponent, ScrolledEventArgs } from '@syncfusion/ej2-angular-lists';
@Component({
  imports: [

    ListViewModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<div class="grid-container"><div><h3>Chat</h3>
    <ejs-listview id='listview' #list [dataSource]='result' height= 320
    width= 400 cssClass='e-list-template' (scroll)="onListScrolled($event)"
  >
    <ng-template #template let-data>
      <div [style.display]="flex" [style.justify-content]="data.positionClass === 'right' ? 'flex-end' : ''" class="e-list-wrapper e-list-multi-line">
        <span style="display: block; white-space: normal; max-width: 80%; padding: 10px; background-color: #e0e0e0; border-radius: 10px; word-wrap: break-word;" class="e-list-item-header">
          {{ data.text }}
        </span>
      </div>
    </ng-template>
  </ejs-listview></div></div>`,
  styles: [`.grid-container {
    display: grid;
    grid-template-columns: repeat(3, 1fr);
    grid-gap: 20px;
    align-items: center;
  }

  h3 {
    margin: 0;
  }

  /* Optional: Add styling to headings and divs */
  .grid-container > div {
    border: 1px solid #ddd;
    padding: 20px;
    border-radius: 5px;
    background-color: #f4f4f4;
  }`],
})
export class AppComponent {
  @ViewChild('list')
```

```

listViewInstance?: ListViewComponent;
//define the array of string
public data: object[] = [{
  text: "Hi Guys, Good morning! \uD83D\uDE0A, I'm very delighted to share
with you the news that our team is going to launch a new mobile
application",
  positionClass: 'right',
}, {
  text: "Oh! That's great \uD83D\uDE42",
  positionClass: 'left',
},
{
  text: 'That is a good news \uD83D\uDE00',
  templateHeight: '80px',
  positionClass: 'right',
},
{
  text: 'What kind of application we are gonna launch? \uD83E\uDD14',
  positionClass: 'left',
},
{
  text: 'A kind of "Emergency Broadcast App" like being able to just
invite someone to teams without it impacting how many people have official
access',
  positionClass: 'right',
},
{
  text: 'Who will be the client users for our applications? ',
  positionClass: 'left',
},
{
  text: 'Is currently the only way to invite someone through 0365? Just
wondering down the road how organization would want to handle that with
freelancers, like being able to just invite someone to teams without it
impacting how many people have official access \uD83D\uDE14',
  positionClass: 'right',
},
{
  text: 'Yes, however, that feature of inviting someone from outside your
organization is planned - expected closer to GA next year \uD83D\uDC4D',
  positionClass: 'left',
},
{
  text: 'I guess we should switch things over to hear for a while. How
long does the trial last? \uD83E\uDD14',
  positionClass: 'right',
},
{ text: 'I think the trial is 30 days. \uD83D\uDE03', positionClass:
'left' },
{
  text: 'Only 0365 only members of your organization. They said that they
are listening to customer feedback and hinted that guest users would be
brought in down the road \uD83D\uDE09',
  positionClass: 'right',
},
{ text: 'Cool thanks! \uD83D\uDC4C', positionClass: 'left' }
];

```

```

public count = 0;
public itemsRendered = 7;
public itemsPerScroll = 5;
public result: { [key: string]: Object; }[] = [{ text: 'text',
positionClass: 'positionClass' }];
ngOnInit() {
    // Initialize the list with the first set of items
    this.result = this.data.slice(0, this.itemsRendered) as { [key: string]:
Object }[];
}
onListScrolled(args: ScrolledEventArgs): void {
    if (args.scrollDirection === 'Bottom' && args.distanceY < 100) {
        if (this.itemsRendered < this.data.length) {
            const startIndex = this.itemsRendered;
            const endIndex = Math.min(this.itemsRendered + this.itemsPerScroll,
this.data.length);
            this.result = this.result.concat(this.data.slice(startIndex,
endIndex) as { [key: string]: Object }[]);
            this.listViewInstance?.addItem(this.result);
            this.itemsRendered = endIndex;
        }
    }
}
}

```

MAIN.TS

```

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

```

Style in Angular ListView component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on user preference.

Customizing ListView

Use the following CSS to customize the ListView.

```

`CSS
.e-listview {
border: 5px solid rgb(173, 255, 47);
}
`

```

Customizing the list items

Use the following CSS to customize the items of ListView.

```

`CSS
.e-listview .e-list-item {
text-align: center;
}
`

```

```
color: pink;
background-color: #2fa1ff;
}
`
```

Customizing ListView's header

Use the following CSS to customize the header of ListView control.

```
`CSS
.e-listview .e-list-header{
color: #2fa1ff;
justify-content: center;
}
`
```

Customizing group header of ListView

Use the following CSS to customize the category of the group items.

```
`CSS
.e-listview .e-list-group-item {
color: rgb(173, 255, 47);
background-color: maroon;
text-align: end;
}
`
```

Customizing the hover state of ListView control

Use the following CSS to customize the list item when hovering.

Customizing ListView hover state with the checkbox checked

```
`CSS
.e-listview .e-list-item.e-hover.e-active.e-checklist {
color: rgb(83, 5, 79);
background-color: rgb(173, 255, 47);
}
`
```

Customizing ListView hover state

```
`CSS
.e-listview .e-list-item.e-hover {
color:red;
background-color: rgb(173, 255, 47);
}
```



```
}
,
```

Customizing selected item of ListView control

Use the following CSS to customize the selected list item.

Customizing ListView's selected item with the checkbox checked

`CSS

```
.e-listview .e-list-item.e-checklist.e-focused.e-active {
color: rgb(83, 5, 79);
background-color: rgb(0, 15, 100);
}
,
```

Customizing ListView's selected item

`CSS

```
.e-listview .e-list-item.e-focused {
color: #2fa1ff;
background-color: rgb(0, 15, 100);
}
,
```

Accessibility in Angular ListView component

The ListView 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 ListView 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 ListView component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the ListView component:

Attributes	Purpose
---	---
<code>role=list</code>	Specifies the non selectable list container.
<code>role=listitem</code>	Specifies the role of each item in a selectable ListView and its containment within the list.
<code>role=presentation</code>	Specifies the role of non selectable list element.
<code>role=checkbox</code>	Indicates checkbox control along with listitem element.
<code>aria-checked</code>	Indicates the current checked state of checkbox.
<code>aria-label</code>	Provides an accessible name for the ListView Checkbox.
<code>aria-disabled</code>	Indicates element is perceivable but disabled.

Keyboard interaction

The ListView 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 ListView component.

Keyboard shortcuts	Actions
--------------------	---------

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

| Arrow Up | Move to the previous list item |

| Arrow Down | Move to the next list item |

| Space | Check and uncheck the targeted list from the whole list |

| BackSpace | Get back to the previous lists if it is nested list |

| Home | Moves focus to first list item |

| End | Moves focus to last list item |

Ensuring accessibility

The ListView component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the ListView component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the ListView component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Ej1 api migration in Angular Listview component

This article describes the API migration process of ListView component from Essential JS 1 to Essential JS 2

| Behaviour | API in Essential JS 1 | API in Essential JS 2 |

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

| Virtualization | **Property:** *allowVirtualScrolling*
<ej-listview [dataSource]="data" [allowVirtualScrolling]='true' [virtualScrollMode]='normal' ></ej-listview> | **Property:** *enableVirtualization*
<ejs-listview [dataSource]='data' [enableVirtualization]='true'></ejs-listview> |

| Checkbox | **Property:** *enableCheckMark*
<ej-listview [dataSource]='data' [enableCheckMark]='true' ></ej-listview> | **Property:** *showCheckBox*
<ejs-listview [dataSource]='data' [showCheckBox]='true'></ejs-listview> |

| Fields | **Property:** *fieldSettings*
<ej-listview [dataSource]="data" [fieldSettings]="fieldsdata" [enableGroupList]="true"></ej-listview>

public fieldsdata = {text: 'textfield'}; | **Property:** *fields*
<ejs-listview [dataSource]='data' [fields]='fieldsdata' ></ejs-listview>

public fieldsdata = {enabled: 'enable_item', groupBy: 'groupByProp'}; |

| Template | **Property:** *renderTemplate*
<ej-listview [dataSource]='data' [renderTemplate]='true'>
 <div>ddd</div>
 </ej-listview > | **Property:** *template*
<ejs-listview [dataSource]='data'>
 <ng-template #template let-data="">
 <div>string template</div>
 </ng-template>
 </ejs-listview> |

| Animation | **Not Applicable** | **Property:** *animation* `
 <ejs-listview [dataSource]='data' [animation]='animation'></ejs-listview>` `
` `
public animation = { effect: 'SlideLeft', duration: 400, easing: 'ease' };|`

| Enable | **Not Applicable** | **Property:** *enable* `
<ejs-listview [dataSource]='data' [enable]="true"></ejs-listview>`|

| Template for grouping | **Not Applicable** | **Property:** *groupTemplate* `
 <ejs-listview [dataSource]='data' [fields]='fieldsdata'>
 <ng-template #groupTemplate let-data="">
 <div>group template</div>
 </ng-template>
 </ejs-listview>` `
` `
public fieldsdata = {groupBy: 'groupByProp'};|`

| Template for header | **Not Applicable** | **Property:** *headerTemplate* `
 <ejs-listview [dataSource]='data' [showHeader]="true">
 <ng-template #headerTemplate let-data="">
 <div>header template</div>
 </ng-template>
 </ejs-listview>`|

| HTML attributes | **Not Applicable** | **Property:** *htmlAttributes* `
 <ejs-listview [dataSource]='data' [htmlAttributes]="attribute"></ejs-listview>``
` `
public attribute = {id: 'listid', class: 'listtest'};|`

| Clear | **Method:** *clear()* `
<ej-listview [dataSource]='data' #listViewInstance></ej-listview>``
` `
this.listViewInstance.widget.clear();|` **Property** *dataSource* `
<ejs-listview [dataSource]='data' #listViewInstance></ejs-listview>``
` `
public data: [key: string]: Object }[] = [];` `
` `this.listViewInstance.destroy();|`

| IsChecked | **Method:** *isChecked()* `
 <ej-listview [dataSource]='data' #listViewInstance [enableCheckMark]='true'></ej-listview>``
` `
this.listViewInstance.widget.isChecked(2);|` **Method:** *checkItem()* `
<ejs-listview [dataSource]='data' [showCheckBox]='true'></ejs-listview>``
` `
public isChecked(li: HTMLLIElement): boolean` `
` `{
return li.classList.contains('e-active');
}`|

| Load Ajax Content | **Method:** *loadAjaxContent()* `
<ej-listview [dataSource]='data' #listViewInstance [enableAjax]='true'></ej-listview>``
` `
this.listViewInstance.widget.loadAjaxContent('load1.html');` | **Event:** *onSuccess* `
<ejs-listview [dataSource]='data' [template]="template"></ejs-listview>` `

public template: string;` `
` `public ajax: Ajax = new Ajax('./template.html', 'GET', false);
` `this.ajax.onSuccess= (e: string)=>{` `
` `this.template = e;
` `}
` `this.ajax.send();|`

| Remove CheckMark | **Method:** *removeCheckMark()* `
 <ej-listview [dataSource]='data' #listViewInstance [enableCheckMark]='true'></ej-listview>``
` `
this.listViewInstance.widget.removeCheckMark(2);|` **Method:** *uncheckItem()* `
<ejs-listview [dataSource]='data' #listViewInstance></ejs-listview>``
` `
this.listViewInstance.uncheckItem ({id:'2'});|`

| Set Active | **Method:** *setActive()* `
 <ej-listview [dataSource]='data' #listViewInstance [persistSelection]='true'></ej-listview>``
` `
this.listViewInstance.widget.setActive(2);|` **Method:** *selectItem()* `
 <ejs-listview [dataSource]='data' #listViewInstance></ejs-listview>``
` `
this.listViewInstance.selectItem({id:'2'});|`

```
| Select | Not Applicable | Event: select <br /><ejs-listview [dataSource]='data'
(select)="onSelect()"></ejs-listview><br/> <br/>public onSelect(){};|
```

```
| Ajax Before Load | Event: ajaxbeforeLoad <br /> <ej-listview [dataSource]='data'
[enableAjax]='true' (ajaxBeforeLoad)= 'ajaxBeforeLoad()'></ej-listview><br/> <br/>public
ajaxBeforeLoad(){}; | Event: onSuccess <br /> <ejs-listview [dataSource]='data'
[template]="template"></ejs-listview> <br/><br/>public template: string;<br /> public ajax: Ajax
= new Ajax('./template.html', 'GET', false);<br /> this.ajax.onSuccess= (e: string)=>{ <br />
this.template = e;<br /> }<br /> this.ajax.send();|
```

```
| Ajax Complete | Event: ajaxComplete <br /> <ej-listview [dataSource]='data'
[enableAjax]='true' (ajaxComplete)= 'ajaxComplete()'></ej-listview><br/> <br/>public
ajaxComplete(){}; | Event: onSuccess <br /> <ejs-listview [dataSource]='data'
[template]="template"></ejs-listview> <br/><br/>public template: string;<br /> public ajax: Ajax
= new Ajax('./template.html', 'GET', false);<br /> this.ajax.onSuccess= (e: string)=>{ <br />
this.template = e;<br /> }<br /> this.ajax.send();|
```

```
| Ajax Error | Event: ajaxError <br /> <ej-listview [dataSource]='data' [enableAjax]='true'
(ajaxError)= 'ajaxError()'></ej-listview><br/> <br/>public ajaxError(){} | Event: onError <br />
<ejs-listview [dataSource]='data' [template]="template"></ejs-listview> <br/><br/>public
template: string;<br /> public ajax: Ajax = new Ajax('./template.html', 'GET', false);<br />
this.ajax.onError= (e: string)=>{ <br /> this.template = e;<br /> }<br /> this.ajax.send();|
```

How To

Add and remove list items from listview in Angular ListView component

You can add or remove list items from the ListView component using the [addItem](#) and [removeItem](#) methods.

Refer to the following steps to add or remove a list item.

- Render the ListView with data source, and use the [template](#) property to append the delete icon

for each list item. Also, bind the click event for the delete icon using the

[actionComplete](#) handler.

- Render the Add Item button, and bind the click event. On the click event handler, pass data with random id to the [addItem](#) method to add a

new list item on clicking the Add Item button.

- Bind the click handler to the delete icon created in step 1. Within the click event, remove the list item by passing the

delete icon list item to

[removeItem](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { ListViewComponent } from '@syncfusion/ej2-angular-lists';
@Component({
  imports: [

    ListViewModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview #list id='sample-list' [dataSource]='data'
[fields]='fields'>
  <ng-template #template let-data="">
    <div class='text-content'> {{data.text}} <span class = 'delete-icon'
(click)="deleteItem($event)"></span> </div>
  </ng-template>
</ejs-listview>
  <button ej2-button id="btn" (click)="addItem()">Add Item</button>`,
})
export class AppComponent {
  @ViewChild('list')
  listViewInstance?: ListViewComponent;
  //define the array of string
  public data: object[] = [{ text: "Hennessey Venom", id: "1", icon:
"delete-icon" },
  { text: "Bugatti Chiron", id: "2", icon: "delete-icon" },
  { text: "Bugatti Veyron Super Sport", id: "3", icon: "delete-icon" },
  { text: "Aston Martin One- 77", id: "4", icon: "delete-icon" },
  { text: "Jaguar XJ220", id: "list-5", icon: "delete-icon" },
  { text: "McLaren P1", id: "6", icon: "delete-icon" }];
  public fields: Object = {text: "text", iconCss: "icon" };
  deleteItem(args: any) {
    args.stopPropagation();
    let liItem = args.target.parentElement.parentElement;
    this.listViewInstance?.removeItem(liItem);
  }
  addItem(){
    let data = {
      text: "Koenigsegg - " + (Math.random() * 1000).toFixed(0),
      id: (Math.random() * 1000).toFixed(0).toString(),
      icon: "delete-icon"
    };
    this.listViewInstance?.addItem([data]);
  }
}

```

MAIN.TS

```

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

```

Create dual list from listview in Angular Listview component

The dual list contains two ListView. This allows you to move list items from one list to another using the client-side

events. This section explains how to integrate the ListView component to achieve dual list.

Use cases

- Stock exchanges of two different countries
- Job applications (skill sets)

Integration of Dual List

Here, two ListView components have been used to display the list items. An ej2-button is used to transfer data between

the ListView, and a textbox is used to achieve the UI of filtering support.

The dual list supports:

- Moving whole data from one list to another.
- Moving selected data from one list to another.
- Filtering the list by using a client-side typed character.

In the ListView component, sorting is enabled using the

[sortOrder](#) property, and

the [select](#) event is triggered

while selecting an item. Here, the select event is triggered to enable and disable button states.

Manipulating data

Moving whole data from the first list to the second list(>>)

- Here, the whole data can be moved from the first ListView to the second by clicking the first button. When clicking the button,

the whole list items are sliced, and `concat` with the second ListView. This button is enabled only when the data source

of the first ListView is not empty.

Moving whole data from the second list to the first list(<<)

- The functionality of the second button is the same as above, and data is transferred from the second list to the first

list. This button is enabled only when the data source of the second ListView is not empty.

Moving selected item from one list to another list (>) and (<)

- The [Select](#) event is triggered

when selecting a list item in the ListView. The selected items can be transferred between two lists. These buttons will be

enabled when selecting an item in lists.

Filtering method

- The filtering method is used to filter list items when typing a character in the text box. In this

method, the [DataManager](#) has been

used to fetch data from the data source and display in ListView.

Sorting

- By using the dual list, list items can be sorted in the ListView component using the

[sortOrder](#) property.

You can enable sorting in one ListView; in the same order, data can be transferred to another ListView.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from "@angular/core";
import { enableRipple } from "@syncfusion/ej2-base";
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
import { ListViewComponent } from "@syncfusion/ej2-angular-lists";
import { ButtonComponent } from "@syncfusion/ej2-angular-buttons"
enableRipple(true);
@Component({
  imports: [

    ListViewModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<div id="text1">
    <input #textbox class="e-input" type="text" id="firstInput"
placeholder="Filter" title="Type in a name" (keyup)="onFirstKeyUp($event)"
/>
    </div>
    <ejs-listview #list1 id='list-1' [dataSource]='firstListData'
[fields]='fields' [sortOrder]='Ascending'
(select)="onFirstListSelect()"></ejs-listview>
    <div id="btn">
      <button ejs-button #btn1 id="firstBtn"
(click)="firstbtnclick()"> >> </button>
      <button ejs-button #btn2 id="secondBtn" [disabled]=true
(click)="secondbtnclick()"> > </button>
      <button ejs-button #btn3 id="thirdBtn" [disabled]=true
(click)="thirdbtnclick()"> < </button>
      <button ejs-button #btn4 id="fourthBtn"
(click)="fourthbtnclick()"> << </button>
```



```

        </div>
        <div id="text2">
            <input #text class="e-input" type="text" id="secondInput"
placeholder="Filter" title="Type in a name" (keyup)="onSecondKeyUp($event)"
/>
        </div>
        <ejs-listview #list2 id='list-2' [dataSource]='secondListData'
[fields]='fields' [sortOrder]='Ascending'
(select)="onSecondListSelect()"></ejs-listview>
    ,
})
export class AppComponent {
    public fields?: Object;
    public firstListData?:any; secondListData: any;
    Ascending: any;
    constructor(){
        this.firstListData = [
            { text: "Hennessey Venom", id: "list-01" },
            { text: "Bugatti Chiron", id: "list-02" },
            { text: "Bugatti Veyron Super Sport", id: "list-03" },
            { text: "SSC Ultimate Aero", id: "list-04" },
            { text: "Koenigsegg CCR", id: "list-05" },
            { text: "McLaren F1", id: "list-06" }
        ];
        this.secondListData = [
            { text: 'Aston Martin One- 77', id: 'list-07' },
            { text: 'Jaguar XJ220', id: 'list-08' },
            { text: 'McLaren P1', id: 'list-09' },
            { text: 'Ferrari LaFerrari', id: 'list-10' },
        ];
        this.fields = { text: "text", id: "id" };
    }
    @ViewChild('list1')firstListObj?: ListViewComponent;
    @ViewChild('list2')secondListObj?: ListViewComponent;
    @ViewChild('btn1')firstBtnObj?: ButtonComponent;
    @ViewChild('btn2')secondBtnObj?: ButtonComponent;
    @ViewChild('btn3')thirdBtnObj?: ButtonComponent;
    @ViewChild('btn4')fourthBtnObj?: ButtonComponent;
    @ViewChild('textbox')textboxEle: any;
    @ViewChild('text')textEle: any;
    ngAfterViewInit(){
        this.firstListData = (this.firstListObj as ListViewComponent |
any).dataSource.slice();
        this.secondListData = (this.secondListObj as ListViewComponent |
any).dataSource.slice();
    }
    //Here, all list items are moved to the second list on clicking move
all button
    firstbtnclick() {
        (this.secondListObj as ListViewComponent).dataSource =
Array.prototype.concat.call((this.firstListObj as
ListViewComponent).dataSource, (this.secondListObj as
ListViewComponent).dataSource);
        this.updateFirstListData();
        this.firstListObj?.removeMultipleItems((this.firstListObj as
ListViewComponent | any).liCollection);
    }
}

```

```

        this.firstListData = this.firstListData.concat((this.firstListObj as
ListViewComponent).dataSource);
        this.secondListData = (this.secondListObj as ListViewComponent |
any).dataSource.slice();
        (this.firstBtnObj as ButtonComponent).disabled = true;
        this.onFirstKeyUp((e: any) => {});
        this.setButtonState();
    }
    //Here, the selected list items are moved to the second list on clicking
move button
    secondbtnclick() {
        let e = this.firstListObj?.getSelectedItems();
        (this.secondListObj as ListViewComponent).dataSource =
Array.prototype.concat.call((this.secondListObj as
ListViewComponent).dataSource, (e as any).data);
        this.firstListObj?.removeItem((e as any).item);
        this.firstListData = (this.firstListObj as
ListViewComponent).dataSource;
        this.secondListData = (this.secondListObj as ListViewComponent |
any).dataSource.slice();
        this.onFirstKeyUp((e: any) => {});
        (this.secondBtnObj as ButtonComponent).disabled = true;
        this.setButtonState();
    }
    //Here, the selected list items are moved to the first list on clicking
move button
    thirdbtnclick () {
        let e = this.secondListObj?.getSelectedItems();
        (this.firstListObj as ListViewComponent).dataSource =
Array.prototype.concat.call((this.firstListObj as
ListViewComponent).dataSource, (e as any).data);
        this.secondListObj?.removeItem((e as any).item);
        this.secondListData = (this.secondListObj as
ListViewComponent).dataSource;
        this.firstListData = (this.firstListObj as ListViewComponent |
any).dataSource.slice();
        this.onSecondKeyUp((e: any) => {});
        (this.thirdBtnObj as ButtonComponent).disabled = true;
        this.setButtonState();
    }
    //Here, all list items are moved to the first list on clicking move all
button
    fourthbtnclick() {
        (this.firstListObj as ListViewComponent).dataSource =
Array.prototype.concat.call((this.firstListObj as
ListViewComponent).dataSource, (this.secondListObj as
ListViewComponent).dataSource);
        this.updateSecondListData();
        this.secondListObj?.removeMultipleItems((this.secondListObj as
ListViewComponent | any).liCollection);
        this.secondListData = this.secondListData.concat((this.secondListObj
as ListViewComponent).dataSource);
        this.firstListData = (this.firstListObj as ListViewComponent |
any).dataSource.slice();
        this.onSecondKeyUp((e: any) => {});
        this.setButtonState();
    }
}

```

```

//Here, the ListView data source is updated to the first list
updateFirstListData() {
    Array.prototype.forEach.call((this.firstListObj as ListViewComponent
| any).liCollection, (list) => {
        this.firstListData.forEach((data: any, index: any) => {
            if (list.innerText.trim() === data.text) {
                delete this.firstListData[index];
            }
        });
    });
    this.textboxEle.nativeElement.value = '';
    let ds: any[] = [];
    this.firstListData.forEach((data: any) => {
        ds.push(data);
    })
    this.firstListData = ds;
}
//Here, the ListView dataSource is updated for the second list
updateSecondListData() {
    Array.prototype.forEach.call((this.secondListObj as
ListViewComponent | any).liCollection, (list) => {
        this.secondListData.forEach((data: any, index: string | number)
=> {
            if (list.innerText.trim() === data.text) {
                delete this.secondListData[index];
            }
        });
    });
    this.textEle.nativeElement.value = '';
    let ds: any = [];
    this.secondListData.forEach((data: any) => {
        ds.push(data);
    })
    this.secondListData = ds;
}
onFirstListSelect() {
    (this.secondBtnObj as ButtonComponent).disabled = false;
}
onSecondListSelect() {
    (this.thirdBtnObj as ButtonComponent).disabled = false;
}
//Here, filtering is handled using the dataManager for the first list
onFirstKeyUp(e: any) {
    let value = this.textboxEle.nativeElement.value;
    let data = new DataManager(this.firstListData).executeLocal(new
Query().where('text', 'startswith', value, true));
    if (!value) {
        (this.firstListObj as ListViewComponent).dataSource =
this.firstListData.slice();
    } else {
        (this.firstListObj as ListViewComponent | any).dataSource =
data;
    }
}
//Here, filtering is handled using the dataManager for the second list
onSecondKeyUp(e: any) {
    let value = this.textEle.nativeElement.value;

```

```

    let data = new DataManager(this.secondListData).executeLocal(new
Query().where('text', 'startswith', value, true));
    if (!value) {
        (this.secondListObj as ListViewComponent).dataSource =
this.secondListData.slice();
    } else {
        (this.secondListObj as ListViewComponent | any).dataSource =
data;
    }
}
//Here, the state of the button is changed
setButtonState() {
    if ((this.firstListObj as ListViewComponent |
any).dataSource.length) {
        (this.firstBtnObj as ButtonComponent).disabled = false;
    } else {
        (this.firstBtnObj as ButtonComponent).disabled = true;
        (this.secondBtnObj as ButtonComponent).disabled = true;
    }
    if ((this.secondListObj as ListViewComponent |
any).dataSource.length) {
        (this.fourthBtnObj as ButtonComponent).disabled = false;
    } else {
        (this.fourthBtnObj as ButtonComponent).disabled = true;
        (this.thirdBtnObj as ButtonComponent).disabled = true;
    }
}
}

```

MAIN.TS

```

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

```

Customize listview as chat window in Angular ListView component

ListView can be customizable as chat window. To achieve that, use ListView [template](#) property and [Avatar](#) component.

- ListView template property is used to showcase the ListView as chat window.
- Avatar component is used to design the image of contact person.

Refer the below template code snippet for Template of chat window.

```

`typescript
<ng-template #template let-data="">
<div class="settings" *ngIf="data.chat!='receiver' then senderTemplate else receiverTemplate "></div>
<ng-template #senderTemplate>
<div id="content">

```

```

<div class="name">{{data.text}}</div>
<div id="info">{{data.contact}}</div>
</div>

<div id="image" *ngIf="data.avatar!=""><span class="e-avatar img1 e-avatar-circle">{{data.avatar}}</span></div>

<div id="image" *ngIf="data.avatar=="><span class="{{data.pic}} img1 e-avatar e-avatar-circle">
</span></div>
</ng-template>

<ng-template #receiverTemplate>

<div id="image2" *ngIf="data.avatar!=""><span class="e-avatar img2 e-avatar-circle">{{data.avatar}}</span></div>

<div id="image2" *ngIf="data.avatar=="><span class="{{data.pic}} img2 e-avatar e-avatar-circle">
</span></div>

<div id="content1">
<div class="name1">{{data.text}}</div>
<div id="info1">{{data.contact}}</div>
</div>
</ng-template>
</ng-template>
,

```

Chat order in template

In ListView template, we have rendered the list items based on receiver and sender information from dataSource of listview.

Adding messages to chat window

- Use textbox to get message from user.
- Add the textbox message to ListView dataSource using [addItem](#) method.

```

`typescript
public btnClick() {
let value = this.textboxEle.nativeElement.value;
this.listObj.addItem([ { text: "Amenda", contact: value, id: "2", avatar: "A", pic: "", chat: "receiver" } ]);
this.textboxEle.nativeElement.value = "";
}
,

```

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [

    ListViewModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <ejs-listview id='List' #list [dataSource]='data' headerTitle="Chat"
    showHeader="true" width="350px" [fields]='fields'>
      <ng-template #template let-data="">
        <div class="settings" *ngIf="data.chat!='receiver' then
senderTemplate else receiverTemplate "></div>
        <ng-template #senderTemplate>
          <div id="content">
            <div class="name">{{data.text}}</div>
            <div id="info">{{data.contact}}</div>
          </div>
          <div id="image" *ngIf="data.avatar!=''"><span class="e-
avatar img1 e-avatar-circle">{{data.avatar}}</span></div>
          <div id="image" *ngIf="data.avatar==''"><span
class="{{data.pic}} img1 e-avatar e-avatar-circle"> </span></div>
        </ng-template>
        <ng-template #receiverTemplate>
          <div id="image2" *ngIf="data.avatar!=''"><span class="e-
avatar img2 e-avatar-circle">{{data.avatar}}</span></div>
          <div id="image2" *ngIf="data.avatar==''"><span
class="{{data.pic}} img2 e-avatar e-avatar-circle"> </span></div>
          <div id="content1">
            <div class="name1">{{data.text}}</div>
            <div id="info1">{{data.contact}}</div>
          </div>
        </ng-template>
      </ng-template>
    </ejs-listview>
    <div style="width: 350px;margin: 0 auto;"><input #textbox id="name"
style="width: 275px" class="e-input" type="text" placeholder="Type your
message" />
    <button ejs-button id="btn" style="float:right"
(click)="btnClick()">Send</button> </div>
  `
})
export class AppComponent {
  @ViewChild('list') listObj: any;
  @ViewChild('textbox') textboxEle: any;
  public data: Object[] = [
    {
      text: "Jenifer",
      contact: "Hi",
      id: "1",
      avatar: "",
      pic: "pic01", chat: "sender"
    }
  ],

```

```

    { text: "Amenda", contact: "Hello", id: "2", avatar: "A", pic: "", chat:
"receiver" },
    {
      text: "Jenifer",
      contact: "What Knid of application going to launch",
      id: "4",
      avatar: "",
      pic: "pic02", chat: "sender"
    },
    {
      text: "Amenda ",
      contact: "A knid of Emergency broadcast App",
      id: "5",
      avatar: "A",
      pic: "", chat: "receiver"
    },
    {
      text: "Jacob",
      contact: "Can you please elaborate",
      id: "6",
      avatar: "",
      pic: "pic04", chat: "sender"
    },
  ];
  public fields = { text: "Name" };
  public btnClick() {
    let value = this.textboxEle.nativeElement.value;
    this.listObj.addItem([{ text: "Amenda", contact: value, id: "2", avatar:
"A", pic: "", chat: "receiver" }]);
    this.textboxEle.nativeElement.value = "";
  }
}

```

MAIN.TS

```

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

```

Create mobile contact layout from listview in Angular Listview component

You can customize the ListView using the [template](#) property. Refer to the following steps to customize ListView as mobile contact view with our [ej2-avatar](#).

- Render the ListView with [dataSource](#) that has avatar data. You can set avatar data as either text or class names. Refer to the following codes.

```

`typescript
let data: any = [
{
text: "Jenifer",

```

```

contact: "(206) 555-985774",
id: "1",
avatar: "",
pic: "pic01"
},
{ text: "Amenda", contact: "(206) 555-3412", id: "2", avatar: "A", pic: "" },
];
`

```

- Set `avatar` classes in ListView template to customize contact icon. In the following codes, medium size avatar has been set using the class name `e-avatar e-avatar-circle` from data source.

```

{% raw %}
`typescript
<ng-template #template let-data="">
<div class="e-list-wrapper e-list-multi-line e-list-avatar">
<span class="e-avatar e-avatar-circle" *ngIf="data.avatar !== "">{{data.avatar}}</span>
<span class="{{data.pic}} e-avatar e-avatar-circle" *ngIf="data.pic !== ""> </span>
<span class="e-list-item-header">{{data.text}}</span>
<span class="e-list-content">{{data.contact}}</span>
</div>
</ng-template>
`
{% endraw %}

```

Avatars can be set in different sizes in avatar classes. To know more about avatar classes, refer to [Avatar](#).

- Sort the contact names using the [sortOrder](#) property of ListView.
- Enable the [showHeader](#) property, and set the [headerTitle](#) as `Contacts`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

```



```

        ListViewModule
    ],
    standalone: true,
    selector: 'my-app',
    template: `
        <div id="sample">
            <ejs-listview id='List' [dataSource]='data' headerTitle='Contacts'
            cssClass='e-list-template' [showHeader]='true' sortOrder='Ascending'>
                <ng-template #template let-data="">
                    <div class="e-list-wrapper e-list-multi-line e-list-avatar">
                        <span class="e-avatar e-avatar-circle"
*ngIf="data.avatar !== ''">{{data.avatar}}</span>
                        <span class="{{data.pic}} e-avatar e-avatar-circle"
*ngIf="data.pic !== ''"> </span>
                        <span class="e-list-item-header">{{data.text}}</span>
                        <span class="e-list-content">{{data.contact}}</span>
                    </div>
                </ng-template>
            </ejs-listview>
        </div>
    `
})
export class AppComponent {
    // Listview datasource with avatar and image source fields
    public data?: { [key: string]: Object; }[] = [
        {
            text: "Jenifer",
            contact: "(206) 555-985774",
            id: "1",
            avatar: "",
            pic: "pic01"
        },
        { text: "Amenda", contact: "(206) 555-3412", id: "2", avatar: "A", pic: "" },
    ],
    {
        text: "Isabella",
        contact: "(206) 555-8122",
        id: "4",
        avatar: "",
        pic: "pic02"
    },
    {
        text: "William ",
        contact: "(206) 555-9482",
        id: "5",
        avatar: "W",
        pic: ""
    },
    {
        text: "Jacob",
        contact: "(71) 555-4848",
        id: "6",
        avatar: "",
        pic: "pic04"
    },
    { text: "Matthew", contact: "(71) 555-7773", id: "7", avatar: "M", pic: "" },
    ],

```

```

    {
      text: "Oliver",
      contact: "(71) 555-5598",
      id: "8",
      avatar: "",
      pic: "pic03"
    },
    {
      text: "Charlotte",
      contact: "(206) 555-1189",
      id: "9",
      avatar: "C",
      pic: ""
    }
  ];
}

```

MAIN.TS

```

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

```

Customize listview with dynamic tags in Angular Listview component

You can customize the ListView items using the [template](#) property. Here, the dynamic tags are added and removed in the list item from another ListView. Refer to the following steps to achieve this.

- Initialize dynamic ListView with required property that holds the tags of parent ListView, and bind the [select](#) event (triggers when the list item is selected), in which you can get and add the selected item value as tags into parent ListView. Refer to the following code sample.

```

`typescript
//Select the event that is is rendered inside dialog for ListView
addTag(e) {
  let listTag = document.createElement('span');
  listTag.className = 'advanced-option';
  let labelElem = document.createElement('span');
  labelElem.className = 'label';
  let deleteElem = document.createElement('span');
  deleteElem.className = 'delete';
  deleteElem.onclick = this.removeTag;
  labelElem.innerHTML = e.text;
  listTag.appendChild(labelElem);
  listTag.appendChild(deleteElem);
}

```

```

let tag = document.createElement('span');
tag.className = 'advanced-option-list';
tag.appendChild(listTag);
this.listViewInstance.element.querySelector('.e-active').appendChild(tag);
}
`

```

- Render the dialog component with empty content and append the created dynamic ListView object to the dialog on [created](#) event.
- Bind the click event for button icon (+) to update the ListView data source with tags, and open the dialog with this dynamic ListView. Refer to the following code sample.

```

`typescript
//Method to hide/show the dialog and update the ListView data source
renderDialog(id) {
if (document.getElementsByClassName('e-popup-open').length != 0) {
this.dialog.hide();
}
else {
this.listObj.dataSource = this.datasources[id];
this.listObj.dataBind();
this.dialog.show();
}
}
`

```

- Bind the click event with added dynamic tags to remove it. Refer to the following code sample.

```

`typescript
//Method to remove the list item
removeTag() {
this.parentNode.parentNode.remove();
}
`

```

APP.COMPONENT.TS

```

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

```

```

import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { DialogModule } from '@syncfusion/ej2-angular-popups'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import{ListViewComponent} from "@syncfusion/ej2-angular-lists";
import{DialogComponent} from "@syncfusion/ej2-angular-popups";
@Component({
imports: [

    ListViewModule,
    DialogModule,
    ButtonModule

],
standalone: true,
selector: 'my-app',
template: `<div id="sample">
    <ejs-listview #list id='templatelist' [dataSource]='data'
[fields]='fields' width=350 >
    <ng-template #template let-data="">
        <div><span class="templatetext">{{data.Name}} </span> <span
class="designationstyle"><button ejs-button id="{{data.Id}}" class="e-but"
iconCss='e-icons e-add-icon' cssClass='e-small e-round'
(click)='onClick($event)'></button></span></div>
        </ng-template>
    </ejs-listview>
    <ejs-dialog id='dialog' #ejDialog width='200px'
[animationSettings]='animation' [visible]='false' showCloseIcon='true'
[position]='position'>
        <ng-template #content>
            <ejs-listview #List id="list" showHeader=true headerTitle='Favorite'
width='200px' [dataSource]='datasource.Brooke' [fields]='fields'
(select)=addTag($event)'></ejs-listview>
        </ng-template>
    </ejs-dialog>
</div>`
})
export class AppComponent {
    @ViewChild('list') listViewInstance?: ListViewComponent;
    @ViewChild('List') listObj?: ListViewComponent;
    @ViewChild('ejDialog') dialog?: DialogComponent;
    //define the array of string
    public data: Object[] = [{ "Id": "Brooke", "Name": "Brooke" },
    { "Id": "Claire", "Name": "Claire" },
    { "Id": "Erik", "Name": "Erik" },
    { "Id": "Grace", "Name": "Grace" },
    { "Id": "Jacob", "Name": "Jacob" } ] as Object[];
    public fields: Object = {text: "Name"};
    public position?: Object;
    public animation: Object = {effect: 'None'};
    public parentNode?: HTMLElement;
    public brookeTag : Object = [{ "id": "list11", "Name": "Discover Music" },
    { "id": "list12", "Name": "Sales and Events" },
    { "id": "list13", "Name": "Categories" },
    { "id": "list14", "Name": "MP3 Albums" },
    { "id": "list15", "Name": "More in Music" },
    ];
    public claireTag : Object = [{ "id": "list21", "Name": "Songs" },

```

```

{ "id": "list22", "Name": "Bestselling Albums" },
{ "id": "list23", "Name": "New Releases" },
{ "id": "list24", "Name": "Bestselling Songs" },
];
public erikTag : Object = [{ "id": "list31", "Name": "Artwork" },
{ "id": "list32", "Name": "Abstract" },
{ "id": "list33", "Name": "Acrylic Mediums" },
{ "id": "list34", "Name": "Creative Acrylic" },
{ "id": "list35", "Name": "Canvas Art" }
];
public graceTag : Object = [{ "id": "list41", "Name": "Rock" },
{ "id": "list42", "Name": "Gospel" },
{ "id": "list43", "Name": "Latin Music" },
{ "id": "list44", "Name": "Jazz" },
];
public jacobTag: Object = [{ "id": "list51", "Name": "100 Albums" },
{ "id": "list52", "Name": "Hip-Hop and R&B Sale" },
{ "id": "list53", "Name": "CD Deals" }
];
public datasource: any = { "Brooke": this.brookeTag, "Claire":
this.claireTag, "Erik": this.erikTag, "Grace": this.graceTag, "Jacob":
this.jacobTag };
ngAfterViewChecked(){
setTimeout(()=>{
    this.position = { X: (document.querySelector('.e-add-icon') as
HTMLElement | any).getBoundingClientRect().left + 50, Y:
(document.querySelector('.e-add-icon') as any).getBoundingClientRect().top -
5 };
},1000);
}
onClick(e: any){
    this.renderDialog(e.currentTarget.id);
}
renderDialog(id: string | number) {
    if (document.getElementsByClassName('e-popup-open').length != 0) {
        (this.dialog as DialogComponent).hide();
    }
    else {
        (this.listObj as ListViewComponent).dataSource =
this.datasource[id];
        this.listObj?.dataBind();
        (this.dialog as DialogComponent).show();
    }
}
addTag(e: any) {
    let listTag = document.createElement('span');
    listTag.className = 'advanced-option';
    let labelElem = document.createElement('span');
    labelElem.className = 'label';
    let deleteElem = document.createElement('span');
    deleteElem.className = 'delete';
    deleteElem.onclick = this.removeTag;
    labelElem.innerHTML = e.text;
    listTag.appendChild(labelElem);
    listTag.appendChild(deleteElem);
    let tag = document.createElement('span');
    tag.className = 'advanced-option-list';

```

```

tag.appendChild(listTag);
    (this.listViewInstance?.element.querySelector('.e-active') as
any).appendChild(tag);
}
removeTag() {
    (this.parentNode as HTMLElement).remove();
}
}

```

MAIN.TS

```

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

```

Filter list items in the listview in Angular Listview component

The filtered data can be displayed in the ListView component depending upon on user inputs using the [DataManager](#). Refer to the following steps to render the ListView with filtered data.

- Render a textbox to get input for filtering data.
- Render ListView with [dataSource](#), and set the [sortOrder](#) property.
- Bind the **keyup** event for textbox to perform filtering operation. To filter list data, pass the list data source to the [DataManager](#), manipulate the data using the [executeLocal](#) method, and then update filtered data as ListView dataSource.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from "@angular/core";
import { ListViewComponent } from "@syncfusion/ej2-angular-lists";
import { enableRipple } from "@syncfusion/ej2-base";
import { DataManager, Query, ODataV4Adaptor } from "@syncfusion/ej2-data";
enableRipple(true);
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<div id="sample">
    <input #textbox class="e-input" type="text" id="textbox"
placeholder="Filter" title="Type in a name" (keyup)=onkeyup($event) />
    <ejs-listview #list id='list' [dataSource]='listData'
[fields]='fields' [sortOrder]='Ascending'></ejs-listview>
    </div>`,
  styles: [ `
    #list {
      box-shadow: 0 1px 4px #ddd;
      border-bottom: 1px solid #ddd;

```

```

}
#sample {
  height: 220px;
  margin: 0 auto;
  display: table;
}
`
})
export class AppComponent {
  public listData: Object = [
    { text: "Hennessey Venom", id: "list-01" },
    { text: "Bugatti Chiron", id: "list-02" },
    { text: "Bugatti Veyron Super Sport", id: "list-03" },
    { text: "SSC Ultimate Aero", id: "list-04" },
    { text: "Koenigsegg CCR", id: "list-05" },
    { text: "McLaren F1", id: "list-06" }
  ];
  public fields: Object = { text: "text", id: "id" };
  @ViewChild('list')
  listObj?: ListViewComponent;
  @ViewChild('textbox')textboxEle: any;
  Ascending: any;
  onkeyup(event: any){
    let value = this.textboxEle.nativeElement.value;
    let data = new DataManager(this.listData).executeLocal(new
Query().where("text", "startswith", value, true));
    if (!value) {
      (this.listObj as ListViewComponent).dataSource = (this.listData as
any).slice();
    } else {
      ((this.listObj as ListViewComponent).dataSource as any) = data;
    }
    this.listObj?.dataBind();
  }
}

```

MAIN.TS

```

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

```

In this demo, data has been filtered with starting character of the list items. You can also filter list items with ending character by passing the `endswith` in [where](#) clause instead of `startswith`.

Get selected items from listview in Angular Listview component

Single or many items can be selected by users in the ListView component. An API is used to get selected items from the list items. This is called as the [getSelectedItems](#) method.

getSelectedItems method

This is used to get the details of the currently selected item from the list items. It returns the [SelectedItem](#) | [SelectedCollection](#)

The `getSelectedItems` method returns the following items from the selected list items.

Return type	Purpose
----- -----	
text	Returns the text of selected item lists
data	Returns the whole data of selected list items, i.e., returns the fields data of selected li.
item	Returns the collections of list items

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from "@angular/core";
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: "my-app",
  template: `
    <ejs-listview
      #listview
      id="sample-list"
      [dataSource]="data"
      [showCheckBox]="true"
      [fields]="fields"
    ></ejs-listview>
    <br />
    <input
      type="button"
      id="btn"
      ejs-button
      value="Get Selected Items"
      (click)="onClick($event)"
    />
    <div #val id="val"></div>
  `
})
export class AppComponent {
  @ViewChild("listview") element: any;
  @ViewChild("val") valEle: any;
  public data: Object = [
    { text: "Hennessey Venom", id: "list-01" },
    { text: "Bugatti Chiron", id: "list-02", isChecked: true },
    { text: "Bugatti Veyron Super Sport", id: "list-03" },
    { text: "SSC Ultimate Aero", id: "list-04", isChecked: true },
    { text: "Koenigsegg CCR", id: "list-05" },
    { text: "McLaren F1", id: "list-06" },
    { text: "Aston Martin One- 77", id: "list-07", isChecked: true },
    { text: "Jaguar XJ220", id: "list-08" }
  ];
  public fields: Object = { id: "id", isChecked: "isChecked" };
  onClick(event: any) {
```



```

let selecteditem = this.element.getSelectedItems();
this.valEle.nativeElement.innerHTML = "";
for (let i = 0; i < selecteditem["data"].length; i++) {
  let listData = document.createElement("p");
  listData.innerHTML =
    "text : " +
    selecteditem["text"][i] +
    " , " +
    "id : " +
    selecteditem["data"][i].id;
  this.valEle.nativeElement.append(listData);
}
}
}

```

MAIN.TS

```

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

```

Hide checkbox in listview in Angular Listview component

The checkbox of the any list item can be hidden by using [htmlAttributes](#) of [fields](#) object. With

the help of [htmlAttributes](#) we can add unique class to each list item that will be rendered from the data source, from the CSS class we can hide the checkbox of the list item.

In this sample, we had hidden the multiple leaf node of nested list. The [e-checkbox-hidden](#) class has been added in the data source where the checkbox needs to be hidden. Refer the below snippet for simple data source.

```

`typescript
{
  'text': 'New York',
  'id': '3002',
  'category': 'USA',
  'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' }
}
`

```

Even though we have hidden the checkbox the functionality will be same for the list item which might affect the [getSelectedItems](#) method. So, to counteract that we will follow certain logic in the [select](#) event. The Logic here is to remove the [e-active](#) class from the other checkbox hidden list item which will be added when we select on that item and retain [e-active](#) on currently selected item.

In this process we will exclude the visible checkbox list items and only consider the hidden checkbox items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from "@angular/core";
import { ListViewComponent } from "@syncfusion/ej2-angular-lists";
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id="sample">
      <ejs-listview #list id='folderCheckbox' [dataSource]='dataSource'
[fields]='fields' [sortOrder]='Ascending' headerTitle='Mixed Leaf Checkbox
Hidden List' [showHeader]=true [showCheckBox]=true
(select)="onSelect($event)"></ejs-listview>
    </div>
  `,
})
export class AppComponent {
  public dataSource: Object = [
    {
      'text': 'Asia',
      'id': '01',
      'category': 'Continent',
      'child': [{
        'text': 'India',
        'id': '1',
        'category': 'Asia',
        'child': [{
          'text': 'Delhi',
          'id': '1001',
          'category': 'India',
          'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        },
        {
          'text': 'Kashmir',
          'id': '1002',
          'category': 'India',
          'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        },
        {
          'text': 'Goa',
          'id': '1003',
          'category': 'India',
          'htmlAttributes': { 'class': 'e-file' },
        },
      ],
    },
    {
      'text': 'China',
      'id': '2',
      'category': 'Asia',
      'child': [{
        'text': 'Zhejiang',

```

```

        'id': '2001',
        'category': 'China',
        'htmlAttributes': { 'class': 'e-file' },
    },
    {
        'text': 'Hunan',
        'id': '2002',
        'category': 'China',
        'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
    },
    {
        'text': 'Shandong',
        'id': '2003',
        'category': 'China',
        'htmlAttributes': { 'class': 'e-file' },
    }
  ]
},
{
  'text': 'North America',
  'id': '02',
  'category': 'Continent',
  'child': [{
    'text': 'USA',
    'id': '3',
    'category': 'North America',
    'child': [{
      'text': 'California',
      'id': '3001',
      'category': 'USA',
      'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
    },
    {
      'text': 'New York',
      'id': '3002',
      'category': 'USA',
      'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
    },
    {
      'text': 'Florida',
      'id': '3003',
      'category': 'USA',
      'htmlAttributes': { 'class': 'e-file' },
    }
  ]
},
{
  'text': 'Canada',
  'id': '4',
  'category': 'North America',
  'child': [{
    'text': 'Ontario',
    'id': '4001',
    'category': 'Canada',
    'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
  },
  {
    'text': 'Alberta',

```

```

        'id': '4002',
        'category': 'Canada',
        'htmlAttributes': { 'class': 'e-file' },
    },
    {
        'text': 'Manitoba',
        'id': '4003',
        'category': 'Canada',
        'htmlAttributes': { 'class': 'e-file' },
    }
]]
},
{
    'text': 'Europe',
    'id': '03',
    'category': 'Continent',
    'child': [{
        'text': 'Germany',
        'id': '5',
        'category': 'Europe',
        'child': [{
            'text': 'Berlin',
            'id': '5001',
            'category': 'Germany',
            'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        },
        {
            'text': 'Bavaria',
            'id': '5002',
            'category': 'Germany',
            'htmlAttributes': { 'class': 'e-file' },
        },
        {
            'text': 'Hesse',
            'id': '5003',
            'category': 'Germany',
            'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        }
    ]
}, {
    'text': 'France',
    'id': '6',
    'category': 'Europe',
    'child': [{
        'text': 'Paris',
        'id': '6001',
        'category': 'France',
        'htmlAttributes': { 'class': 'e-file' },
    },
    {
        'text': 'Lyon',
        'id': '6002',
        'category': 'France',
        'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
    },
    {
        'text': 'Marseille',
        'id': '6003',
    }
}

```

```

        'category': 'France',
        'htmlAttributes': { 'class': 'e-file' },
    ]]
    ],
    {
        'text': 'Australia',
        'id': '04',
        'category': 'Continent',
        'child': [{
            'text': 'Australia',
            'id': '7',
            'category': 'Australia',
            'child': [{
                'text': 'Sydney',
                'id': '7001',
                'category': 'Australia',
                'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
            },
            {
                'text': 'Melbourne',
                'id': '7002',
                'category': 'Australia',
                'htmlAttributes': { 'class': 'e-file' },
            },
            {
                'text': 'Brisbane',
                'id': '7003',
                'category': 'Australia',
                'htmlAttributes': { 'class': 'e-file' },
            }
        ]
    }, {
        'text': 'New Zealand',
        'id': '8',
        'category': 'Australia',
        'child': [{
            'text': 'Milford Sound',
            'id': '8001',
            'category': 'New Zealand',
            'htmlAttributes': { 'class': 'e-file' },
        },
        {
            'text': 'Tongariro National Park',
            'id': '8002',
            'category': 'New Zealand',
            'htmlAttributes': { 'class': 'e-file' },
        },
        {
            'text': 'Fiordland National Park',
            'id': '8003',
            'category': 'New Zealand',
            'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        }
    ]
    ],
    {
        'text': 'Africa',

```

```

        'id': '05',
        'category': 'Continent',
        'child': [{
            'text': 'Morocco',
            'id': '9',
            'category': 'Africa',
            'child': [{
                'text': 'Rabat',
                'id': '9001',
                'category': 'Morocco',
                'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
            },
            {
                'text': 'Toubkal',
                'id': '9002',
                'category': 'Morocco',
                'htmlAttributes': { 'class': 'e-file' },
            },
            {
                'text': 'Todgha Gorge',
                'id': '9003',
                'category': 'Morocco',
                'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
            }
        ]
    }, {
        'text': 'South Africa',
        'id': '10',
        'category': 'Africa',
        'child': [{
            'text': 'Cape Town',
            'id': '10001',
            'category': 'South Africa',
            'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        },
        {
            'text': 'Pretoria',
            'id': '10002',
            'category': 'South Africa',
            'htmlAttributes': { 'class': 'e-file e-checkbox-hidden' },
        },
        {
            'text': 'Bloemfontein',
            'id': '10003',
            'category': 'South Africa',
            'htmlAttributes': { 'class': 'e-file' },
        }
    ]
    ]
}

];
public fields = { tooltip: 'text' };
@ViewChild('list') listViewInstance?: ListViewComponent;
Ascending: any;
onSelect(args: any){
    let normalElements: HTMLElement[] =
Array.prototype.slice.call((this.listViewInstance as
any).curUL.getElementsByClassName('e-checkbox-hidden'));

```

```

// Looping through all the selected element and removing e-active
class
// to avoid behaviour interference with getSelectedItems method
normalElements.forEach((element) => {
  element.classList.remove('e-active');
});
// Finally adding e-active class to currently selected item except
checkbox item.
// because if it is checkbox item their actions will taken care from
the source side itself.
if (args.item.classList.contains('e-checkbox-hidden')) {
  args.item.classList.add('e-active');
}
}
}

```

MAIN.TS

```

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

```

List items count in group header in Angular ListView component

The ListView component supports wrapping list items into a group based on the category. The category of each list item can be mapped with groupBy field of the data source. You can display grouped list items count in the list-header using the group header template. Refer to the following code sample to display grouped list item count.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='List' [dataSource]='dataSource'
cssClass='e-list-template' [fields]='fields'>
  <ng-template #template let-dataSource="">
    <div class="e-list-wrapper e-list-multi-line e-list-avatar">
      <img class="e-avatar e-avatar-circle" src={{dataSource.image}}
style="background:#BCBCBC" />
      <span class="e-list-item-header">{{dataSource.Name}}</span>
      <span class="e-list-content">{{dataSource.contact}}</span>
    </div>
  </ng-template>
  <ng-template #groupTemplate let-dataSource="">
    <div>

```

```

        <span className="category">{{ dataSource.items[0].category
    }}</span>
        <<span id="count"> {{ dataSource.items.length }} Item(s)</span>
    </div>
</ng-template>
</ejs-listview>`
    })
    export class AppComponent {
        public dataSource: Object = [
            { Name: 'Nancy', contact: '(206) 555-985774', id: '1', image:
'https://ej2.syncfusion.com/demos/src/grid/images/1.png', category:
'Experience'},
            { Name: 'Janet', contact: '(206) 555-3412', id: '2', image:
'https://ej2.syncfusion.com/demos/src/grid/images/3.png', category:
'Fresher' },
            { Name: 'Margaret', contact: '(206) 555-8122', id: '4', image:
'https://ej2.syncfusion.com/demos/src/grid/images/4.png', category:
'Experience' },
            { Name: 'Andrew ', contact: '(206) 555-9482', id: '5', image:
'https://ej2.syncfusion.com/demos/src/grid/images/2.png', category:
'Experience'},
            { Name: 'Steven', contact: '(71) 555-4848', id: '6', image:
'https://ej2.syncfusion.com/demos/src/grid/images/5.png', category:
'Fresher' },
            { Name: 'Michael', contact: '(71) 555-7773', id: '7', image:
'https://ej2.syncfusion.com/demos/src/grid/images/6.png', category:
'Experience' },
            { Name: 'Robert', contact: '(71) 555-5598', id: '8', image:
'https://ej2.syncfusion.com/demos/src/grid/images/7.png', category:
'Fresher' },
            { Name: 'Laura', contact: '(206) 555-1189', id: '9', image:
'https://ej2.syncfusion.com/demos/src/grid/images/8.png', category:
'Experience' },
        ];
        public fields: Object = { text: 'Name', groupBy: 'category' };
    }

```

MAIN.TS

```

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

```

Manipulate listview as grid layout in Angular Listview component

In ListView, list items can be rendered in grid layout with following data manipulations.

- Add Item
- Remove Item
- Sort Items
- Filter Items

Grid Layout

In this section, we will discuss about rendering of list items in grid layout.

- Initialize and render ListView with dataSource which will render list items in list layout.
- Now, add the below CSS to list item. This will make list items to render in grid layout

```
`css
element .e-list-item {
height: 100px;
width: 100px;
float: left;
}
`
```

In the below sample, we have rendered List items in grid layout.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <ejs-listview id='element' [dataSource]='data'>
      <ng-template #template let-data="">
        
      </ng-template>
    </ejs-listview>
  `,
})
export class AppComponent {
  public data = [1, 2, 3, 4, 5, 6, 7, 8, 9];
}
```

MAIN.TS

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

Data manipulation

In this section, we will discuss about ListView data manipulations.

Add Item

We can add list item using [addItem](#) API. This will accept array of data as argument.

```
`typescript
this.$refs.listViewInstance.addItem([{text: 'Apricot', id: '32'}]);
`
```

In the below sample, you can add new fruit item by clicking add button which will open dialog box with fruit name and image URL text box. After entering the item details, click the add button. This will add your new fruit item.

Remove item

We can remove list item using [removeItem](#) API. This will accept fields with `id` or list item element as argument.

```
`typescript
this.$refs.listViewInstance.removeItem({id: '32'});
`
```

In the below sample, you can remove fruit by hovering the fruit item which will show delete button and click that delete button to delete that fruit from your list.

Sort Items

ListView can be sorted either in Ascending or Descending order. To enable sorting in your ListView, set [sortOrder](#) as `Ascending` or `Descending`.

```
`typescript
<ejs-listview sortOrder='Ascending'></ejs-listview>
`
```

We can also set sorting after component initialization.

```
`typescript
this.listViewInstance.sortOrder = 'Ascending'
`
```

In the below sample, we have sorted fruits in `Ascending` order. To sort it in descending, click on sort order icon and vice versa.

Filter Items

ListView data can be filtered with the help of [dataManager](#). After filtering the data, update ListView [dataSource](#) with filtered data.

```
`typescript
let value = this.textboxEle.nativeElement.value; //input text box value
let filteredData = new DataManager(this.listdata).executeLocal(
  new Query().where("text", "startswith", value, true)
);
```

```
listViewInstance.dataSource = filteredData;
```

In the below sample, we can filter fruit items with the help of search text box. This will filter fruit items based on your input. Here we used `startswith` of input text to filter data in DataManager.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { DialogComponent, DialogModule } from '@syncfusion/ej2-angular-popups'
import { Component, ViewChild, ElementRef } from '@angular/core';
import { closest, enableRipple } from '@syncfusion/ej2-base';
import { DataManager, Query } from '@syncfusion/ej2-data';
enableRipple(true);
@Component({
  imports: [
    ListViewModule,
    DialogModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id="sample">
      <div class="headerContainer">
        <div class="e-input-group">
          <input id="search" #searchEle class="e-input" type="text"
placeholder="Search fruits" (keyup)='onKeyUp($event)' />
          <span class="e-input-group-icon e-input-search"></span>
        </div>
        <button ejs-button id="sort" class="e-control e-btn e-small e-
round e-primary e-icon-btn" (click)='sortItems($event)' title="Sort fruits"
data-ripple="true">
          <span class="e-btn-icon e-icons e-sort-icon-
ascending"></span>
        </button>
        <button ejs-button id="add" class="e-control e-btn e-small e-
round e-primary e-icon-btn" (click)='addItem($event)' title="Add fruit"
data-ripple="true">
          <span class="e-btn-icon e-icons e-add-icon"></span>
        </button>
        <ejs-dialog id="dialog" #dialogObj width='300px'
[visible]='false' header='Add Fruit' showCloseIcon='true'
[buttons]='addButtons'>
          <ng-template #content let-data="">
            <div id="listDialog"><div class="input_name"><label
for="name">Fruit Name: </label><input id="name" class="e-input" type="text"
placeholder="Enter fruit name"/></div><div><label for="imgurl">Fruit Image:
</label><input id="imgurl" class="e-input" type="text" placeholder="Enter
image url"/></div></div>
          </ng-template>
        </ejs-dialog>
      </div>
    </div>
```

```

        <ejs-listview id='element' #listview [dataSource]='fruitsdata'
sortOrder='Ascending' (actionComplete)='wireEvents()'>
        <ng-template #template let-data="">
            <div class="fruits"><div class="first"><img id="listImage"
src={{data.imageUrl}} alt="fruit" /><button ejs-button class="delete e-control
e-btn e-small e-round e-delete-btn e-primary e-icon-btn" data-
ripple="true"><span class="e-btn-icon e-icons delete-
icon"></span></button></div><div class="fruitName">{{data.text}}</div></div>
            </ng-template>
        </ejs-listview>
    </div>
    `),
    })
    export class AppComponent {
        @ViewChild('listview') listViewInstance: any;
        @ViewChild('dialogObj') dialogObj: any;
        @ViewChild('searchEle') searchEle: any;
        public fruitsdata = [
            { text: 'Date', id: '1', imageUrl: './dates.jpg' },
            { text: 'Fig', id: '2', imageUrl: './fig.jpg' },
            { text: 'Apple', id: '3', imageUrl: './apple.png' },
            { text: 'Apricot', id: '4', imageUrl: './apricot.jpg' },
            { text: 'Grape', id: '5', imageUrl: './grape.jpg' },
            { text: 'Strawberry', id: '6', imageUrl: './strawberry.jpg' },
            { text: 'Pineapple', id: '7', imageUrl: './pineapple.jpg' },
            { text: 'Melon', id: '8', imageUrl: './melon.jpg' },
            { text: 'Lemon', id: '9', imageUrl: './lemon.jpg' },
            { text: 'Cherry', id: '10', imageUrl: './cherry.jpg' },
        ];
        public addButtons= [{
            click: this.dlgButtonClick.bind(this), buttonModel: { content:
'Add', isPrimary: true }
        }];
        wireEvents() {
            Array.prototype.forEach.call(document.getElementsByClassName('e-delete-
btn'), (ele) => {
                ele.addEventListener('click', this.onDeleteBtnClick.bind(this));
            });
        }
        addItem(args: any) {
            (document.getElementById("name") as HTMLElement | any).value = "";
            (document.getElementById("imgurl") as HTMLElement | any).value = "";
            this.dialogObj.show();
        }
        sortItems(args: any) {
            let ele = (document.getElementById("sort") as
HTMLElement).firstElementChild;
            let des = (ele as Element).classList.contains('e-sort-icon-descending')
? true : false;
            if (des) {
                (ele as Element).classList.remove('e-sort-icon-descending');
                (ele as Element).classList.add('e-sort-icon-ascending');
                this.listViewInstance.sortOrder = 'Ascending';
            } else {
                (ele as Element).classList.remove('e-sort-icon-ascending');
                (ele as Element).classList.add('e-sort-icon-descending');
                this.listViewInstance.sortOrder = 'Descending'

```

```

    }
    this.listViewInstance?.dataBind();
    this.wireEvents();
  }
  onKeyUp(e: any) {
    let value = this.searchEle.nativeElement.value;
    let data = new DataManager(this.fruitsdata).executeLocal(
      new Query().where("text", "startswith", value, true)
    );
    if (!value) {
      this.listViewInstance.dataSource = this.fruitsdata.slice();
    } else {
      this.listViewInstance.dataSource = data;
      this.listViewInstance?.dataBind();
    }
  }
  onDeleteBtnClick(e: any) {
    e.stopPropagation();
    let li = closest(e.currentTarget, '.e-list-item');
    let data = this.listViewInstance?.findItem(li);
    this.listViewInstance?.removeItem(data);
    new DataManager(this.fruitsdata).remove('id', { id: data.id });
  }
  dlgButtonClick() {
    let name = (document.getElementById("name") as HTMLElement | any).value;
    let url = (document.getElementById("imgurl") as HTMLElement |
any).value;
    let id = Math.random() * 10000;
    this.listViewInstance?.addItem([{ text: name, id: id, imgUrl: url }]);
    this.fruitsdata.push({ text: name, id: 'id', imgUrl: url });
    this.listViewInstance?.element.querySelector('[data-uid="'+ id +
'"]')?.getElementsByClassName('e-delete-btn')[0].addEventListener('click',
this.onDeleteBtnClick.bind(this));
    this.dialogObj.hide();
  }
}

```

MAIN.TS

```

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

```

Render listview with hyper link navigation in Angular Listview component

We can use `anchor` tag along with `href` attribute in our ListView `template` property for navigation.

`typescript

```

<ng-template #template let-data="">
<a target='_blank' href="{{data.url}}">{{data.name}}</a>
</ng-template>

```

In the below sample, we have rendered **ListView** with search engines URL.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <div id="sample">
      <ejs-listview id='List' [dataSource]='data' headerTitle='Search engines'
showHeader='true'>
        <ng-template #template let-data="">
          <a target='_blank' href="{{data.url}}">{{data.name}}</a>
        </ng-template>
      </ejs-listview>
    </div>
  `,
})
export class AppComponent {
  public data=[
    {name: 'Google', url: 'https://www.google.com'},
    {name: 'Bing', url: 'https://www.bing.com' },
    {name: 'Yahoo', url: 'https://www.yahoo.com'},
    {name: 'Ask.com', url: 'https://www.ask.com'},
    {name: 'AOL.com', url: 'https://www.aol.com'},
  ];
}
```

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 list items in Angular Listview component

In **ListView** component, we don't have drag and drop support. But we can achieve this requirement using [TreeView](#) component with **ListView** appearance.

Drag and Drop in **TreeView** component was enabled by setting [allowDragAndDrop](#) to **true**.

`typescript

```
<ejs-treeview id='element' [fields]='fields' allowDragAndDrop='true'></ejs-treeview>
```

`

The TreeView component is used to represent hierarchical data in a tree like structure. So, list items in TreeView can be dropped to child of target element. we can prevent this behaviour by cancelling the [nodeDragStop](#) and [nodeDragging](#) events.

`typescript

```
<ejs-treeview id='element' [fields]='fields' allowDragAndDrop='true'
(nodeDragging)='onDragStop($event)' (nodeDragStop)='onDragStop($event)'></ejs-treeview>

fields= { dataSource: this.data, id: 'id', text: 'text' },
onDragStop(args) {
//Block the Child Drop operation in TreeView
let draggingItem = document.getElementsByClassName("e-drop-in");
if (draggingItem.length == 1) {
draggingItem[0].classList.add('e-no-drop');
args.cancel = true;
}
}
,`
```

In the below sample, we have rendered draggable list items.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { TreeViewModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
@Component({
imports: [

    TreeViewModule
],
standalone: true,
selector: 'my-app',
template: `
    <div id="sample">
        <ejs-treeview id='element' [fields]='fields' allowDragAndDrop='true'
(nodeDragging)='onDragStop($event)'
(nodeDragStop)='onDragStop($event)'></ejs-treeview>
    </div>
    `
})
export class AppComponent {
    public data=[
        { text: 'Hennessey Venom', id: 'list-01' },
        { text: 'Bugatti Chiron', id: 'list-02' },
        { text: 'Bugatti Veyron Super Sport', id: 'list-03' },
        { text: 'SSC Ultimate Aero', id: 'list-04' },
        { text: 'Koenigsegg CCR', id: 'list-05' },
        { text: 'McLaren F1', id: 'list-06' },
    ],
```

```

    { text: 'Aston Martin One- 77', id: 'list-07' },
    { text: 'Jaguar XJ220', id: 'list-08' },
    { text: 'McLaren P1', id: 'list-09' },
    { text: 'Ferrari LaFerrari', id: 'list-10' },
  ];
  public fields = { dataSource: this.data, id: 'id', text: 'text' };
  onDragStop(args: any) {
    //Block the Child Drop operation in TreeView
    let draggingItem = document.getElementsByClassName("e-drop-in");
    if (draggingItem.length == 1) {
      draggingItem[0].classList.add('e-no-drop');
      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));

```

Load html content via ajax in Angular Listview component

We can set external HTML page content as [template](#) for our **ListView** component by making use of **AJAX** call.

`typescript

```
let ajax = new Ajax('./template.html', 'GET', false);
```

```
ajax.onSuccess = (e)=>{
```

```
this.listtemplate = e;
```

```
};
```

```
ajax.send();
```

```
,
```

In the below sample, we have rendered smartphone settings template from external **HTML** file.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild } from '@angular/core';
import { ListViewComponent } from '@syncfusion/ej2-angular-lists';
import { Ajax } from '@syncfusion/ej2-base';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,

```



```

        selector: 'my-app',
        template: `
            <ejs-listview id='List' [dataSource]='data' [fields]='fields'
            showHeader='true' headerTitle='Settings' [template]="listtemplate">
                </ejs-listview>
        `
    })
    export class AppComponent {
        public listtemplate?: ListViewComponent;
        public data = [
            { name: 'Network & Internet', id: '0', description: 'Wi-Fi, mobile, data usage, hotspot' },
            { name: 'Connected devices', id: '1', description: 'Bluetooth, cast, NFC' },
            { name: 'Battery', id: '2', description: '18% -4h 12m left' },
            { name: 'Display', id: '3', description: 'Wallpaper, sleep, font size' },
            { name: 'Sound', id: '4', description: 'Volume, vibration, Do Not Disturb' },
            { name: 'Storage', id: '5', description: '52% used - 15.48 GB free' }
        ];
        public fields: Object = {text: 'name', id: 'id'};
        ngOnInit(){
            let ajax = new Ajax('./template.html', 'GET', false);
            ajax.onSuccess = (e: any)=>{
                this.listtemplate = e;
            };
            ajax.send();
        }
    }

```

MAIN.TS

```

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

```

Load list items in child list dynamically in Angular Listview component

To load list items in child list dynamically, push the new list item data into the existing [dataSource](#) using the `[select]`..(<https://ej2.syncfusion.com/angular/documentation/api/list-view/#select>) event.

Refer to the following steps to load list item into the child list:

1. Initially, render the ListView with the required data source.
2. Bind the [select](#) event that triggers selecting list item in the ListView component. By using the select event, you can push the new list item to the child list of the data source on specifying its item index. Item index can be obtained from the [SelectEventArgs](#) of the select event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'

```

```

import { Component, ViewEncapsulation } from '@angular/core';
import { SelectEventArgs } from '@syncfusion/ej2-lists';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='listview' [dataSource]='dataSource'
[template]='wintemplate' headerTitle='Folders' [showHeader]='true'
[showIcon]='true' [fields]='fields' (select)='onSelect($event)'></ejs-
listview>
`,
  styles: [`
#listview {
display: block;
max-width: 400px;
margin: auto;
border: 1px solid #dddddd;
border-radius: 3px;
}
#listview.e-listview .e-list-icon {
height: 24px;
width: 30px;
}
.folder, .file {
background:
url('http://ej2.syncfusion.com/demos/src/listview/images/file_icons.png')
no-repeat;
background-size: 300%;
}
.folder{
background-position: -5px -461px;
}
.file {
background-position: -5px -151px;
}
.list {
color:deeppink !important;
}
`],
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public dataSource: { [key: string]: Object }[] = [
    {
      id: '01', text: 'Music', icon: 'folder',
      child: [
        { id: '01-01', text: 'Gouttes.mp3', icon: 'file' }
      ]
    },
    {
      id: '02', text: 'Videos', icon: 'folder',
      child: [
        { id: '02-01', text: 'Naturals.mp4', icon: 'file' },
        { id: '02-02', text: 'Wild.mpeg', icon: 'file' },
      ]
    }
  ]
}

```

```

    ],
    {
        id: '03', text: 'Documents', icon: 'folder',
        child: [
            { id: '03-01', text: 'Environment Pollution.docx', icon: 'file' },
            { id: '03-02', text: 'Global Water, Sanitation, & Hygiene.docx', icon: 'file' },
            { id: '03-03', text: 'Global Warming.ppt', icon: 'file' },
            { id: '03-04', text: 'Social Network.pdf', icon: 'file' },
            { id: '03-05', text: 'Youth Empowerment.pdf', icon: 'file' },
        ]
    },
    {
        id: '04', text: 'Pictures', icon: 'folder',
        child: [
            {
                id: '04-01', text: 'Camera Roll', icon: 'folder',
                child: [
                    { id: '04-01-01', text: 'WIN_20160726_094117.JPG', icon: 'file' },
                    { id: '04-01-02', text: 'WIN_20160726_094118.JPG', icon: 'file' },
                    { id: '04-01-03', text: 'WIN_20160726_094119.JPG', icon: 'file' }
                ]
            },
            {
                id: '04-02', text: 'Wind.jpg', icon: 'file'
            },
            {
                id: '04-02', text: 'Stone.jpg', icon: 'file'
            },
            {
                id: '04-02', text: 'Home.jpg', icon: 'file'
            },
            {
                id: '04-02', text: 'Bridge.png', icon: 'file'
            }
        ]
    },
    {
        id: '05', text: 'Downloads', icon: 'folder',
        child: [
            { id: '05-01', text: 'UI-Guide.pdf', icon: 'file' },
            { id: '05-02', text: 'Tutorials.zip', icon: 'file' },
            { id: '05-03', text: 'Game.exe', icon: 'file' },
            { id: '05-04', text: 'TypeScript.7z', icon: 'file' },
        ]
    },
];

public fields: Object = { iconCss: 'icon', tooltip: 'text' };
wintemplate: any;
//Select event to add new list item in child page
onSelect(args: SelectEventArgs) {
    //Add new file to the child page of selected list item

```

```
//Add new file to the child page of selected list item
(this.dataSource[args.index]['child'] as any).push({ id: '01-02', text:
'Newly Added File', icon: 'file', htmlAttributes: { role: 'li', class:
'list' } });
}
}
```

MAIN.TS

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

Display spinner until list items are loaded in Angular Listview component

The features of the ListView component such as remote data-binding take more time to fetch data from corresponding dataSource/remote URL. In this case, you can use EJ2 [Spinner](#) to enhance the appearance of the UI. This section explains how to load a spinner component to groom the appearance.

Refer to the following code sample to render the spinner component.

```
`typescript
createSpinner({
target: this.spinnerEle.nativeElement
});
showSpinner(this.spinnerEle.nativeElement);
`
```

Refer to the following code sample to render the ListView component.

```
`typescript
let listViewInstance: ListView = new ListView({
//Bind the DataManager instance to the dataSource property
dataSource= new DataManager({
url: '//js.syncfusion.com/ejServices/Wcf/Northwind.svc/',
crossDomain: true
}),
//Bind the Query instance to the query property
query= new Query().from('Products').select('ProductID,ProductName').take(10),
//Map the appropriate columns to the fields property
fields= { id: 'ProductID', text: 'ProductName' },
});
//Render the initialized ListView
```

```
listviewInstance.appendTo("#element");
```

Here, the data is fetched from **Northwind** Service URL; it takes a few seconds to load the data. To enhance the UI, the spinner component has been rendered initially. After the data is loaded from remote URL, the spinner component will be hidden in ListView [actionComplete](#) event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild, ViewEncapsulation } from "@angular/core";
import { DataManager, Query, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { createSpinner, showSpinner, setSpinner } from '@syncfusion/ej2-angular-popups';
@Component({
  imports: [
    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `
    <ejs-listview id='element' #list [dataSource]='dataSource' [width]='300'
    [query]='query' [fields]='fields' [showHeader]='true'
    [headerTitle]='headertitle' (actionComplete)='onActionComplete($event)' >
    </ejs-listview>
    <div #spinner id="spinner" ></div>
  `,
  styles: [ `
    #element {
      display: block;
      max-width: 400px;
      min-height: 200px;
      margin: auto;
      border: 1px solid #dddddd;
      border-radius: 3px;
    }
  ` ]
})
export class AppComponent {
  @ViewChild('spinner') spinnerEle:any;
  public dataSource= new DataManager({
    url: '//js.syncfusion.com/ejServices/Wcf/Northwind.svc/',
    crossDomain: true
  });
  public query = new
  Query().from('Products').select('ProductID,ProductName').take(10);
  public fields: Object = { id: 'ProductID', text: 'ProductName' };
  public headertitle = 'Product Name';
  ngAfterViewInit() {
    createSpinner({
      target: this.spinnerEle.nativeElement
    });
    showSpinner(this.spinnerEle.nativeElement);
  }
}
```

```

    }
    onActionComplete(args:any) {
        this.spinnerEle.nativeElement.style.display = "none";
    }
}

```

MAIN.TS

```

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

```

Trace all events in listview in Angular Listview component

The ListView component triggers events based on its actions. The events can be used as extension points to perform custom operations. Refer to the following steps to trace the ListView events:

1. Render the ListView with [dataSource](#), and bind the [actionBegin](#), [actionComplete](#), and [select](#) events.
2. Perform custom operations in [actionBegin](#), [actionComplete](#), and [select](#) events.
3. Provide event log details for [actionBegin](#) and [actionComplete](#) events, and they will be displayed in the event trace panel when the ListView action starts and the dataSource bound successfully.
4. Get the selected item details from the [SelectEventArgs](#) in the [select](#) event, and display the selected list item text in the event trace panel while selecting list items.
5. Use clear button to remove event trace information.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { SelectEventArgs } from '@syncfusion/ej2-lists';
@Component({
  imports: [

    ListViewModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<div id="sample">
    <div class="content-wrapper">
      <ejs-listview id='listview-def' [dataSource]='listData' [width]='250'
        (select)='onSelect($event)' (actionBegin)='onActionBegin($event)'
        (actionComplete)='onActionComplete($event)'></ejs-listview>
    </div>
    <div id="list_event">
      <h4><b>Event Trace</b></h4>
      <div id="evt">
        <div class="eventarea" style="height:273px;overflow:
auto">

```

```

        <span #EventLog class="EventLog" id="EventLog"
style="word-break: normal;"></span>
    </div>
    <div class="evtbtn">
        <input ej-button id="clear" type="button"
value="Clear" (click)='onclick($event)'>
    </div>
</div>
</div>
</div>`,
    styles: [ `
    #EventLog b {
    color: #388e3c;
}
#listview-def {
    border: 1px solid #dcdcdc;
}
.content-wrapper {
    padding-left: 40px;
    padding-top: 36px;
}
.evtbtn {
    margin-top: 40px;
    margin-left: 70px;
}
/* csslint ignore:start */
hr {
    margin-top: 6px !important;
    margin-bottom: 6px !important;
}
/* csslint ignore:end */
#evt {
    border: 1px solid #dcdcdc;
    padding: 10px;
    min-width: 10px;
}
#sample {
    display: inline-flex;
}
.eventarea {
    min-width: 250px;
}
#list_event {
    margin-top: -25px;
    padding-left: 40px;
    min-width: 200px;
}
    `],
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    @ViewChild('EventLog') EventLogEle:any;
    public listData: Object = [
    { text: "Hennessey Venom", id: "list-01" },
    { text: "Bugatti Chiron", id: "list-02" },
    { text: "Bugatti Veyron Super Sport", id: "list-03" },
    { text: "SSC Ultimate Aero", id: "list-04" },

```

```

    { text: "Koenigsegg CCR", id: "list-05" },
    { text: "McLaren F1", id: "list-06" },
    { text: "Aston Martin One- 77", id: "list-07" },
    { text: "Jaguar XJ220", id: "list-08" },
    { text: "McLaren P1", id: "list-09" },
    { text: "Ferrari LaFerrari", id: "list-10" }
  ];
  onclick(event: any) {
    this.EventLogEle.nativeElement.innerHTML = "";
  }
  onSelect(args: SelectEventArgs) {
    this.appendElement(args.text + "<b>&#160;&#160;is selected</b><hr>");
  }
  onActionBegin(args: any) {
    this.appendElement("<b>actionBegin </b> event is triggered<hr>");
  }
  onActionComplete(args: any) {
    this.appendElement("<b>actionComplete</b> is triggered <hr>");
  }
  appendElement(html: string): void {
    let span: HTMLElement = document.createElement("span");
    span.innerHTML = html;
    let log: HTMLElement = this.EventLogEle.nativeElement;
    log.insertBefore(span, log.firstChild);
  }
}

```

MAIN.TS

```

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

```

Use dynamic templates in listview based on device in Angular Listview component

The Syncfusion Essential JS2 components are desktop and mobile-friendly. So, you can use Syncfusion components in both modes. The component templates are not always fixed. Applications may need to load various templates depending upon the device.

Integration

In the ListView component, template support is being used. In some cases, the component wrapper is always responsive across all devices, but the template contents are dynamically changed with unspecified (sample side) dimensions. CSS customization is also needed in sample-side to align template content responsively in both mobile and desktop modes. Here, two templates have been loaded for mobile and desktop modes. To check the device mode, a **browser module** has been imported from the **ej2-base** package.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { Component } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';

```



```

@Component({
  imports: [

    ListViewModule

  ],
  standalone: true,
  selector: 'my-app',
  template: `<ejs-listview id='List' [dataSource]='dataSource'
[template]='templatecheck ? mob_template : win_template'
headerTitle='Syncfusion Blog' [showHeader]='true'>
    <ng-template #mob_template let-datasource="">
      <div class="settings">
        <div id="postContainer">
          <div id="postImg">
            <img src={{dataSource.image}} /></div>
          <div id="content">
            <div id="info">
              <div id="logo">
                <div id="share">
                  <span class="share"></span> </div>
                  <div id="comments"> <span class="comments"></span>
</div>
                  <div id="bookmark"> <span class="bookmark"></span>
</div>
                </div>
              </div>
            <div class="name">{{dataSource.Name}}</div>
            <div class="description">{{dataSource.content}}</div>
            <div class="timeStamp">{{dataSource.timeStamp}} </div>
          </div>
        </div>
      </div>
    </ng-template>
    <ng-template #win_template let-datasource="">
      <div class="settings">
        <div id="postContainer">
          <div id="postImg">
            <img src={{dataSource.image}} /></div>
          <div id="content">
            <div class="name">{{dataSource.Name}}</div>
            <div class="description">{{dataSource.content}}</div>
            <div id="info">
              <div id="logo">
                <div id="share">
                  <span class="share"></span> </div>
                  <div id="comments"> <span class="comments"></span>
</div>
                  <div id="bookmark"> <span class="bookmark"></span>
</div>
                </div>
              <div class="timeStamp">{{dataSource.timeStamp}} </div>
            </div>
          </div>
        </div>
      </div>
    </ng-template></ejs-listview>`
  })

```

```

export class AppComponent {
  //Define an array of JSON data
  public dataSource: any = [
    { Name: 'IBM Open-Sources Web Sphere Liberty Code', content: 'In September, IBM announced that it would be open-sourcing the code for WebSphere...', id: '1', image: 'https://ej2.syncfusion.com/demos/src/listview/images/1.png', timeStamp: 'Syncfusion Blog - October 19, 2017' },
    { Name: 'Must Reads: 5 Big Data E-books to upend your development', content: 'Our first e-book was published in May 2012-jQuery Succinctly was the start of over...', id: '2', image: 'https://ej2.syncfusion.com/demos/src/listview/images/2.png', timeStamp: 'Syncfusion Blog - October 18, 2017' },
    { Name: 'The Syncfusion Global License: Your Questions, Answered ', content: 'Syncfusion recently hosted a webinar to cover the ins and outs of the Syncfusion global...', id: '4', image: 'https://ej2.syncfusion.com/demos/src/listview/images/3.png', timeStamp: 'Syncfusion Blog - October 18, 2017' },
    { Name: 'Know: What is Coming from Microsoft this Fall ', content: 'On October 17, Microsoft will release its Fall Creators Update for the Windows 10 platform...', id: '5', image: 'https://ej2.syncfusion.com/demos/src/listview/images/6.png', timeStamp: 'Syncfusion Blog - October 17, 2017' }
  ];
  public fields: Object = { text: 'Name' };
  public templatecheck?:boolean;
  constructor() {
    this.templatecheck = Browser.isDevice;
  }
}

```

MAIN.TS

```

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

```

Integrate pager component with listview in Angular Listview component

The first and foremost step is to obtain the **Pager** component from **Grid**. Install the ej2-angular-grids package using the following command.

```
`shell
```

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

```
,
```

Import the Pager to the ListView sample which has been created.

```
`shell
```

```
import { Pager } from "@syncfusion/ej2-angular-grids";
```

```
,
```

The [totalRecordsCount](#) property of the Pager must be specified whenever using this particular component. By using [pageSize](#) property, the number of list items to be displayed is made available. The [pageCount](#) property allows the user to specify the visibility of the page numbers accordingly. Since the paging sample in the upcoming code snippet uses these three properties, the explanation provided here are minimal and to the point. For further API concerns in Pager component, [click here](#).

With the help of the [query](#) property of ListView, the user can specify the number of records to be displayed in the current page.

The [query](#) property helps in splitting the entire datasource based on the user's convenience. In the sample provided below, when clicking the next button in pager, it fetches the datasource based on the page size and the current page of the Pager component.

The [headerTemplate](#) and the [template](#) property of ListView is defined within ng-template. The required styles can be changed here accordingly.

```
`typescript
public clickevent(args) {
this.query = new Query().range((args.currentPage - 1) this.pageSize, (args.currentPage this.pageSize));
}
`
```

In the above code snippet, the event stores the [currentPage](#) value, and the datasource which is to be displayed in the next page is obtained.

Note: When [Link to the Video](#) isn't mentioned, it defaults to 12 records per page.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { PagerModule } from '@syncfusion/ej2-angular-grids'
import { Component } from '@angular/core';
import { Browser } from '@syncfusion/ej2-base';
import { datasource } from './datasource';
import { DataManager, Query, JsonAdaptor } from '@syncfusion/ej2-data';
import { Pager } from '@syncfusion/ej2-angular-grids';
@Component({
  imports: [
    ListViewModule, PagerModule, ButtonModule
  ],
  standalone: true,
  selector: 'my-app',
  template: `<div class="control-section">
    <ejs-listview id='listview' [dataSource]='data' [query]='query'
    showHeader='true' >
      <ng-template #headerTemplate let-data="">
        <table class="w-100"> <tr><td class="w-25">Order ID</td><td class="w-
45">Ship      Name</td><td class="w-25">Ship City</td></tr></table>
      </ng-template>
      <ng-template #template let-data="">
```

```

        <table class="w-100"> <tr><td class="w-25">{{data.OrderID}}</td><td
class="w-45">{{data.ShipName}}</td><td class="w-25"
>{{data.ShipCity}}</td></tr></table>
    </ng-template>
</ejs-listview>
<ejs-pager [pageSize]= 'pagesize' [totalRecordsCount]='totalrec'
[pageCount]='pageCount' (click)='clickevent($event)'>
</ejs-pager>
</div>`
    ))
export class AppComponent {
public totalrec: number = datasource.length;
public pageSize: number = 10;
public pageCount: number = 2;
//Define an array of JSON data
public data: Object = new DataManager({
    json: datasource,
    adaptor: new JsonAdaptor
});
public query = new Query().range(0, this.pageSize);
templatecheck: boolean;
public clickevent(args: any) {
    this.query = new Query().range((args.currentPage - 1) * this.pageSize,
(args.currentPage * this.pageSize));
}
constructor() {
    this.templatecheck = Browser.isDevice;
}
}

```

MAIN.TS

```

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

```

Maps

Getting started with Angular Maps component

This section explains you the steps required to create a map and demonstrate the basic usage of the maps component.

You can explore some useful features in the Maps component using the following video.

Dependencies

The following is a list of the dependencies required to use the Maps component.

```

`javascript
|-- @syncfusion/ej2-angular-maps
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-angular-maps

```

```
|-- @syncfusion/ej2-maps
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-svg-base
|-- @syncfusion/ej2-data
\
```

Setup Angular Environment

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

To install Angular CLI use the following command.

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

Create an Angular Application

Start a new Angular application using below Angular CLI command.

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

Installing Syncfusion Maps package

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

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

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

Ivy library distribution package

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

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

```
`bash
npm install @syncfusion/ej2-angular-maps --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-maps@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-maps@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-maps:"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 Maps Module

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

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

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

```
`javascript
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-container',
  // specifies the template string for the maps component

```

```

template: <ejs-maps id='maps-container'></ejs-maps>,
encapsulation: ViewEncapsulation.None
})
export class AppComponent { }
`

```

Add the **world-map** data in the **app.component.ts** file.

Note: Refer the data for [world-map](#) here. These data must be imported in the **src/app/app.component.ts** file.

```

`javascript
import { world_map } from './world-map';
`

```

Bind the **world-map** data to the **shapeData** property of the **layer** in the Maps control.

```

`typescript
@Component({
// specifies the template string for the maps component
template: `<ejs-maps id='maps-container'>
<e-layers>
<e-layer [shapeData] = 'shapeData'></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent {
public shapeData: object = world_map;
}
`

```

```

<!-- markdownlint-disable MD033 -->

```

Now use the `<app-container>` in the **index.html** instead of default one.

```

`html
<app-container></app-container>
`

```

```

`typescript
@Component({
selector: 'app-container'

```

```

  })
  ,

```

- Now run the application in the browser using the below command.

```

,

```

```

npm start

```

```

,

```

The below example shows a basic map.

```

`typescript
import { Component } from '@angular/core';
import { world_map } from './world-map';
@Component({
  selector: 'app-container',
  // specifies the template string for the maps component
  template: `<ejs-maps id='maps-container'>
    <e-layers>
    <e-layer [shapeData] = 'shapeData'></e-layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent {
  public shapeData: object = world_map;
}
,

```

Module Injection

Maps component are segregated into individual feature-wise modules. In order to use a particular feature,

you need to inject its feature module using `Maps.Inject()` method. Find the modules available in maps and its description as follows.

- `AnnotationsService` - Inject this provider to use annotations feature.
- `BubbleService` - Inject this provider to use bubble feature.
- `DataLabelService` - Inject this provider to use data label feature.
- `HighlightService` - Inject this provider to use highlight feature.
- `LegendService` - Inject this provider to use legend feature.
- `MarkerService` - Inject this provider to use marker feature.

- `MapsTooltipService` - Inject this provider to use tooltip series.
- `NavigationLineService` - Inject this provider to use navigation lines feature.
- `SelectionService` - Inject this provider to use selection feature.
- `ZoomService` - Inject this provider to use zooming and panning feature.

For this application we are going to use tooltip, data label and legend features of the maps.

Now import the MapsTooltip, DataLabel and Legend modules from maps package

`@syncfusion/ej2-angular-maps`

```
`javascript
```

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

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

```
import { AppComponent } from './app.component';
```

```
import { MapsComponent, LegendService, DataLabelService, MapsTooltipService } from '@syncfusion/ej2-angular-maps';
```

```
@NgModule({
```

```
  imports: [
```

```
    BrowserModule,
```

```
  ],
```

```
  declarations: [AppComponent, MapsComponent],
```

```
  bootstrap: [AppComponent],
```

```
  providers: [ MapsComponent, LegendService, DataLabelService, MapsTooltipService ]
```

```
})
```

```
,
```

[Render shapes from GeoJSON data](#)

This section explains how to bind GeoJSON data to the map.

```
`javascript
```

```
let usMap: object =
```

```
{
```

```
  "type": "FeatureCollection",
```

```
  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
```

```
  "features": [
```

```
    { "type": "Feature", "properties": { "iso31662": "MA", "name": "Massachusetts", "admin": "United States of America" }, "geometry": {
```

```
      "type": "MultiPolygon",
```

```
      "coordinates": [ [ [ [ -70.801756294617277, 41.248076234530558 ] ] ] ] }
```

```

}
]
//..
};
`

```

Elements in the maps will get rendered in the layers. So add a layer collection to the maps by using `[layers]` property. Now bind the GeoJSON data to the `[shapeData]` property.

`[app.module.ts]`

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
  }
}

```

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 MD022 -->

[Bind data source to map](#)

<!-- markdownlint-disable MD009 -->

The following properties in layers are used for binding data source to map.

- `dataSource`

- `shapeDataPath`
- `shapePropertyPath`

The [dataSource](#) property takes collection value as input. For example, the list of objects can be provided as input. This data is further used in tooltip, data label, bubble, legend and in color mapping.

The [shapeDataPath](#) property used to refer the data ID in dataSource. Where as, the [shapePropertyPath](#) property is used to refer the column name in shapeData to identify the shape. Both the properties are related to each other. When the values of the shapeDataPath property in the dataSource property and the value of shapePropertyPath in the shapeData property match, then the associated object from the dataSource is bound to the corresponding shape.

The JSON object "electionData" is used as data source below.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
        'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
        'dataSource'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  ngOnInit(): void {
    this.dataSource = [{ "Country": "China", "Membership":
    "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
      { "Country": "Poland", "Membership": "Non-Permanent" },
      { "Country": "Sweden", "Membership": "Non-Permanent" }];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
  };
}
```

MAIN.TS

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

Apply Color Mapping

The Color Mapping feature supports customization of shape colors based on the underlying value of shape received from bounded data. Specify the field name from which the values have to be compared for the shapes in [colorValuePath](#) property in [shapeSettings](#).

Specify color and value in [colorMapping](#) property. Here '#D84444' is specified for 'Trump' and '#316DB5' is specified for 'Clinton'.

[app.module.ts]

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-maps id='rn-container' >
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  ngOnInit(): void {
    this.dataSource = [{ "Country": "China", "Membership":
"Permanent"},
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent"},
      { "Country": "Kazakhstan", "Membership": "Non-Permanent"},
      { "Country": "Poland", "Membership": "Non-Permanent"},
      { "Country": "Sweden", "Membership": "Non-Permanent"}];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
```

```

        colorValuePath: 'Membership',
        colorMapping: [
          {
            value: 'Permanent', color: '#D84444'
          },
          {
            value: 'Non-Permanent', color: '#316DB5'
          }
        ]
      };
    }
  }
}

```

MAIN.TS

```

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

```

Add Title for Maps

You can add a title using [titleSettings](#) property to the map to provide quick information to the user about the shapes rendered in the map.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [titleSettings] = 'titleSettings' >
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
        'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
        'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public titleSettings?: object;
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  ngOnInit(): void {
    this.titleSettings = {
      text: 'World map membership',
    }
  }
}

```

```

        titleStyle: {
            size: '16px'
        }
    }
    this.dataSource = [{ "Country": "China", "Membership":
"Permanent"},
        { "Country": "France", "Membership": "Permanent" },
        { "Country": "Russia", "Membership": "Permanent"},
        { "Country": "Kazakhstan", "Membership": "Non-Permanent"},
        { "Country": "Poland", "Membership": "Non-Permanent"},
        { "Country": "Sweden", "Membership": "Non-Permanent"}];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
            {
                value: 'Permanent', color: '#D84444'
            },
            {
                value: 'Non-Permanent', color: '#316DB5'
            }
        ]
    };
}
}

```

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

You can show legend for the maps by setting true to the [visible](#) property in [legendSettings](#) object and by injecting the [LegendService](#) module using [@NgModule.providers](#) method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>

```

```

    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [{ "Country": "China", "Membership":
"Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
      { "Country": "Poland", "Membership": "Non-Permanent" },
      { "Country": "Sweden", "Membership": "Non-Permanent" } ];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
      colorValuePath: 'Membership',
      colorMapping: [
        {
          value: 'Permanent', color: '#D84444'
        },
        {
          value: 'Non-Permanent', color: '#316DB5'
        }
      ]
    };
    this.legendSettings = {
      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));

```

Add Data Label

You can add data labels to show additional information of the shapes in map. This can be achieved by setting [visible](#) property to true in the [dataLabelSettings](#) object and by injecting `DataLabelService` module using `@NgModule.providers` method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [DataLabelService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
    [dataLabelSettings] = 'dataLabelSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {
      visible: true,
      labelPath: 'name',
      smartLabelMode: 'Trim'
    };
  }
}

```

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 useful when you cannot display information by using the data labels due to space constraints.

You can enable tooltip by setting the [visible](#) property as true in [tooltipSettings](#) object and by injecting [MapsTooltipService](#) module using `@NgModule.providers` method.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```



```

import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService, DataLabelService } from '@syncfusion/ej2-
angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService, DataLabelService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-maps id='rn-container'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
[dataLabelSettings] = 'dataLabelSettings'[tooltipSettings] =
'tooltipSettings'></e-layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeSettings?: object;
  public tooltipSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {
      visible: true,
      labelPath: 'name',
      smartLabelMode: 'Trim'
    };
    this.tooltipSettings = {
      visible: true,
      valuePath: 'name'
    };
  }
}

```

MAIN.TS

```

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

```

[Populate data in Angular Maps component](#)[Geometry types](#)

GeoJSON data contains geometry objects with properties such as geometry types and coordinates. The geometry types are the values present in the geometry objects of the GeoJSON data that specify the

type of shape to be rendered, as well as the coordinates that help to draw the shape's boundary line. The supportive geometry types are:

```
| Shapes | Supported |
| --- | --- |
| Polygon | Yes |
| MultiPolygon | Yes |
| LineString | Yes |
| MultiLineString | Yes |
| Point | Yes |
| MultiPoint | Yes |
| GeometryCollection | Yes |
```

Shape data

The shape data collection describes geographical shape information that is available in GeoJSON format. The Map shapes are rendered with this data. The custom shapes such as seat selection in bus, seat selection in a cricket stadium and more useful information can be also added as [shapeData](#) in the layer of the Maps.

```
`typescript
```

```
export let usMap = // Paste all the content copied from the JSON file.
```

```
,
```

Data source

The [dataSource](#) property is used to represent statistical data in the Maps component, and it accepts a collection of values as input. For example, a list of objects as input can be provided to the data source. This data source will be used to color the map, display data labels, and display tooltip, among other things.

The data source is populated with JSON data relative to shape data and stored as JSON object. In the below example, **populationData** can be used as data source in Maps.

```
`typescript
```

```
export let populationData: object[] = [
{
'code': 'AF',
'value': 53,
'name': 'Afghanistan',
'population': 29863010,
'density': 119
},
{
```

```
'code': 'AL',  
'value': 117,  
'name': 'Albania',  
'population': 3195000,  
'density': 111  
,  
{  
'code': 'DZ',  
'value': 15,  
'name': 'Algeria',  
'population': 34895000,  
'density': 15  
,  
{  
'code': 'AO',  
'value': 15,  
'name': 'Angola',  
'population': 18498000,  
'density': 15  
,  
{  
'code': 'AR',  
'value': 15,  
'name': 'Argentina',  
'population': 40091359,  
'density': 14  
,  
{  
'code': 'AM',  
'value': 109,  
'name': 'Armenia',  
'population': 3230100,  
'density': 108
```

```

}
];
`

```

Data binding

The following properties in the [layers](#) are used for binding data in the Maps component. Both the properties are related to each other.

- `shapePropertyPath`
- `shapeDataPath`

shapePropertyPath

The [shapePropertyPath](#) property is used to refer the field name in the [shapeData](#) property of shape layers to identify the shape. When the values of [shapeDataPath](#) property from the [dataSource](#) property and [shapePropertyPath](#) property from the [shapeData](#) property match, then the associated object from the data source is bound to the corresponding shape.

`world-map.ts` file contains following data and its field **name** value is used to map the corresponding shape with the provided data source.

```

`typescript
export let world_map: object = {
  "type": "Feature",
  "properties": {
    "admin": "Afghanistan",
    "name": "Afghanistan",
    "continent": "Asia"
  },
  "geometry": { "type": "Polygon", "coordinates": [[[61.21081709172573,
https://ej2.syncfusion.com/angular/documentation. ],
...
`

```

shapeDataPath

The [shapeDataPath](#) property is similar to the [shapePropertyPath](#) property, but it refers to the field name in the [dataSource](#) property. For example, [populationData](#) contains the **code**, **value**, **name**, **population** and **density** fields. Here, the **name** field is set to the `shapeDataPath` to map the corresponding value of field name in shape data.

In the below example, both **name** fields contain the same value as **Afghanistan**, this value is matched in both shape data and data source, so that the details associated with **Afghanistan** will be mapped to the corresponding shape and used to color the corresponding shape, display data labels, display tooltips, and more.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings]='shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      {
        'code': 'AF',
        'value': 53,
        'name': 'Afghanistan',
        'population': 29863010,
        'density': 119,
        'color': '#DEEBAE'
      },
      {
        'code': 'AL',
        'value': 117,
        'name': 'Albania',
        'population': 3195000,
        'density': 111,
        'color': '#A4D6AD'
      },
      {
        'code': 'DZ',
        'value': 15,
        'name': 'Algeria',
        'population': 34895000,
        'density': 15,
        'color': '#37AFAB'
      },
      {
        'code': 'AO',
        'value': 15,
        'name': 'Angola',
        'population': 18498000,

```

```

        'density': 15,
        'color': '#547C84'
    },
    {
        'code': 'AR',
        'value': 15,
        'name': 'Argentina',
        'population': 40091359,
        'density': 14,
        'color': '#CEBF93'
    },
    {
        'code': 'AM',
        'value': 109,
        'name': 'Armenia',
        'population': 3230100,
        'density': 108,
        'color': '#a69d70'
    }
]
this.shapeData = world_map;
this.shapePropertyPath = 'name';
this.shapeDataPath = 'name';
this.shapeSettings = {
    colorValuePath: 'color',
    fill: '#E5E5E5'
}
};
}

```

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 MD010 -->

Binding complex data source

Data from data source can be bind to the Maps in two different ways.

1. Bind the field name directly to the properties as [shapeDataPath](#), [colorValuePath](#), [valuePath](#) and [shapeValuePath](#).
2. Bind the field name as `data.field` to the properties as [shapeDataPath](#), [colorValuePath](#), [valuePath](#) and [shapeValuePath](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, MapsTooltipService, BubbleService } from
 '@syncfusion/ej2-angular-maps'

```

```

import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService, BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath'
          [shapeSettings]='shapeSettings' [dataSource] = 'dataSource'
[tooltipSettings]='tooltipSettings'
          [bubbleSettings]='bubbleSettings'
[markerSettings]='markerSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public tooltipSettings?: object;
  public bubbleSettings?: object[];
  public markerSettings?: object[];
  ngOnInit(): void {
    this.dataSource = [
      { "Continent": "North America", 'color': '#71B081',
        data: { "continent": "North America", 'color': '#71B081' } },
      { "Continent": "South America", 'color': '#5A9A77',
        data: { "continent": "South America", 'color': '#5A9A77' } },
      { "Continent": "Africa", 'color': '#498770',
        data: { "continent": "Africa", 'color': '#498770' } },
      { "Continent": "Europe", 'color': '#39776C',
        data: { "continent": "Europe", 'color': '#39776C' } },
      { "Continent": "Asia", 'color': '#266665',
        data: { "continent": "Asia", 'color': '#266665' } },
      { "Continent": "Australia", 'color': '#124F5E',
        data: { "continent": "Australia", 'color': '#124F5E' } }
    ]
    this.shapeData = world_map;
    this.shapePropertyPath = 'continent';
    this.shapeDataPath = 'data.continent';
    this.shapeSettings = {
      colorValuePath : 'data.color'
    };
  }
}

```

```

        this.tooltipSettings = {
            visible: true,
            valuePath: 'data.continent'
        };
        this.bubbleSettings = [
            {
                visible: true,
                valuePath: 'data.value',
                colorValuePath: 'data.color',
                animationDuration: 0,
                minRadius: 20,
                maxRadius: 90,
                opacity: 0.8,
                dataSource: [
                    { 'name': 'India', 'value': 18.89685398845257,
'population': 391292635,
                    data: { 'color': 'red', 'population': 391292635,
'value': 189685398845257 }
                }
                ],
                tooltipSettings: {
                    visible: true,
                    valuePath: 'data.population',
                    template: "<div>${data.population}</div>"
                }
            }
        ];
        this.markerSettings = [
            {
                visible: true,
                dataSource: [
                    { latitude: 37.6276571, longitude: -122.4276688, name:
'San Bruno',
                        data: { x: 37.6276571, y: -122.4276688, name: 'San
Bruno', shape: 'Pentagon',
                            color: 'red', imageUrl: 'images/ballon.png' }
                    },
                    { latitude: 33.5302186, longitude: -117.7418381, name:
'Laguna Niguel',
                        data: { x: 33.5302186, y: -117.7418381, name: 'Laguna
Niguel', color: 'blue',
                            shape: 'Pentagon', imageUrl: 'images/ballon.png' }
                    }
                ],
                shapeValuePath: "data.shape",
                colorValuePath: "data.color",
                height: 20,
                width: 20,
                offset: {
                    y: -10,
                    x: 0
                },
                longitudeValuePath: "data.y",
                latitudeValuePath: "data.x",
                tooltipSettings: {
                    visible: true,
                    valuePath: 'data.name',

```



```

        format: "${data.name}: ${data.x} : ${data.y}"
      },
      animationDuration: 0
    }
  ]
};
}

```

MAIN.TS

```

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

```

Layers in Angular Maps component

The Maps component is rendered through [layers](#) and any number of layers can be added to the Maps.

Multilayer

The Multilayer support allows loading multiple shape files and map providers in a single container, enabling Maps to display more information. The shape layer or map providers are the main layers of the Maps. Multiple layers can be added as **SubLayer** over the main layers using the [type](#) property of [layers](#).

Sublayer

Sublayer is a type of shape file layer. It allows loading multiple shape files in a single map view. For example, a sublayer can be added over the main layer to view geographic features such as rivers, valleys and cities in a map of a country. Similar to the main layer, elements in the Maps such as markers, bubbles, color mapping and legends can be added to the sub-layer.

In this example, the United States map shape is used as shape data by utilizing **usa.ts** file, and **texas.ts** and **california.ts** files are used as sub-layers in the United States map.

[app.component.ts]

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
import { california } from './california';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData' [shapeSettings] = 'shapeSettings'></e-
layer>
        <e-layer [shapeData] = 'shapeData1' [shapeSettings] = 'shapeSettings1'
[type] = 'type'></e-layer>
    `
})

```

```

    </e-layers>
    </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public shapeData?: object;
    public shapeSettings?: object;
    public shapeData1?: object;
    public type?: string;
    public shapeSettings1?: object;
    ngOnInit(): void {
      this.shapeData = usa_map;
      this.shapeSettings = {
        fill: '#E5E5E5',
        border: {
          color: 'black',
          width: 0.1
        }
      }
      this.shapeData1 = california;
      this.type = 'SubLayer',
      this.shapeSettings1 = {
        fill: 'rgba(141, 206, 255, 0.6)',
        border: {
          color: '#1a9cff',
          width: 0.25
        }
      }
    }
  }
}

```

MAIN.TS

```

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

```

[app.module.ts]

```

`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the MapsModule for the Maps component
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { AppComponent } from './app.component';
@NgModule({
  // declaration of ej2-angular-maps module into NgModule
  imports: [ BrowserModule, MapsModule ],
  declarations: [ AppComponent ],

```

```
bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Displaying different layer in the view

Multiple shape files and map providers can be loaded simultaneously in Maps. The [baseLayerIndex](#) property is used to determine which layer on the user interface should be displayed. This property is used for the Maps drill-down feature, so when the [baseLayerIndex](#) value is changed, the corresponding shape is loaded. In this example, two layers can be loaded with the World map and the United States map. Based on the given [baseLayerIndex](#) value the corresponding shape will be loaded in the user interface. If the [baseLayerIndex](#) value is set to **0**, then the world map will be loaded.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
import { world_map } from './world-map';
import { california } from './california';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [baseLayerIndex] = 'baseLayerIndex'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData'></e-layer>
        <e-layer [shapeData] = 'shapeData1'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeData1?: object;
  public baseLayerIndex?: number;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeData1 = usa_map;
    this.baseLayerIndex = 1;
  }
}
```

MAIN.TS

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

Rendering custom shapes

Custom shapes (also known as custom maps) can be rendered in Maps to represent bus seat booking, cricket stadium, basic home plan/sketch, and so on. To accomplish this, a JSON file in GeoJSON format with proper geometries must be created manually or with the assistance of any online map vendor. The GeoJSON file created must be set to the [shapeData](#) in the Maps layer, and the [geometryType](#) must be set as **Normal**.

Please refer this [link](#) for an example GeoJSON file containing information about bus seat selection.

Please refer this [link](#) for more information and a live demonstration.

Providers

Map provider in Angular Maps component

The OpenStreetMap (OSM) is the online Maps provider built by a community of developers; it is free to use under an open license. It allows to view geographical data in a collaborative way from anywhere on the earth. The OSM Maps provides small tile images based on our requests and combines those images into a single image to display the Maps area in the Maps component.

Adding OpenStreetMap

The OSM Maps can be rendered using the [urlTemplate](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style='display:block'>
      <e-layers>
        <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public urlTemplate?: string;
  ngOnInit(): void {
    this.urlTemplate =
    'https://tile.openstreetmap.org/level/tileX/tileY.png';
  }
}
```

MAIN.TS

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

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

Changing the tile server of the OpenStreetMap

The OSM tile server can be changed by setting the tile URL in the [urlTemplate](#) property. For more details about the OSM tile server, refer [here](#).

Enabling zooming and panning

The OSM Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a Maps for in-depth analysis. Panning helps to move a Maps around to focus the targeted area.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style='display:block'
    [zoomSettings]='zoomSettings'>
      <e-layers>
        <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public zoomSettings?: object;
  public urlTemplate?: string;
  ngOnInit(): void {
    this.urlTemplate =
      'https://tile.openstreetmap.org/level/tileX/tileY.png';
    this.zoomSettings = {
      enable: true,
      toolbars: ["Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset"]
    }
  }
}
```

MAIN.TS

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

Adding markers and navigation line

Markers can be added to the layers of OSM Maps by setting the corresponding location's coordinates of latitude and longitude using [markerSettings](#). Navigation lines can be added on top of an OSM Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [navigationLineSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, NavigationLineService, ZoomService } from
 '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, NavigationLineService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

```

        {
            latitude: 40.724546,
            longitude: -73.850344,
            name: "New York"
        }
    ]
    });
    this.navigationLineSettings = [{
        visible: true,
        color: "blue",
        width: 5,
        angle: 0.1,
        latitude: [34.060620, 40.724546],
        longitude: [-118.330491, -73.850344]
    }];
}
}

```

MAIN.TS

```

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

```

Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the OSM Maps layer for highlighting a particular continent or country in OSM Maps by adding another layer and specifying the [type](#) property of Maps layer to **SubLayer**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { africa_continent } from './africa-continent';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style='display:block'>
      <e-layers>
        <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
        <e-layer [type] = 'type' [shapeData]='shapeData'
        [shapeSettings]='shapeSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public type?: string;
  public shapeData?: object;
}

```

```

public shapeSettings?: object;
public urlTemplate?: string;
ngOnInit(): void {
    this.urlTemplate =
'https://tile.openstreetmap.org/level/tileX/tileY.png';
    this.type = 'SubLayer';
    this.shapeData = africa_continent;
    this.shapeSettings = {
        fill: 'blue'
    };
}
}

```

MAIN.TS

```

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

```

Enabling legend

The legend can be added to the tile Maps by setting the [visible](#) property of [legendSettings](#) to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { markerDataSource } from './markerdata';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps
  id="rn-container"
  style="display:block"
  [legendSettings]="legendSettings"
>
  <e-layers>
    <e-layer [urlTemplate]= 'urlTemplate'
      [shapePropertyPath]="shapePropertyPath"
      [shapeDataPath]="shapeDataPath"
      [markerSettings]="markerSettings"
      [shapeSettings]="shapeSettings"
    ></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeDataPath?: string;

```



```

public shapePropertyPath?: string;
public shapeSettings?: object;
public markerSettings?: object;
public legendSettings?: object;
public urlTemplate?: string;
ngOnInit(): void {
  this.urlTemplate =
'https://tile.openstreetmap.org/level/tileX/tileY.png';
  this.shapeDataPath = 'name';
  this.shapePropertyPath = 'name';
  this.legendSettings = {
    visible: true,
    type: 'Markers',
    useMarkerShape: true,
    toggleLegendSettings: {
      enable: true,
      applyShapeSettings: false,
      border: {
        color: 'green',
        width: 2,
      },
    },
  };
  this.shapeSettings = {
    fill: '#E5E5E5',
  };
  this.markerSettings = [
    {
      dataSource: markerDataSource,
      colorValuePath: 'color',
      shapeValuePath: 'shape',
      legendText: 'country',
      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));

```

Bing maps in Angular Maps component

Bing Maps is a online Maps provider, owned by Microsoft. As like OSM, it provide Maps tile images based on our requests and combines those images into a single one to display Maps area.

Adding Bing Maps

The Bing Maps can be rendered using the [urlTemplate](#) property, which is based on the URL generated by the [getBingUrlTemplate](#) method in the Maps. The format of the required URL of Bing Maps varies from other online map providers. As a result, a built-in [getBingUrlTemplate](#) method has been included that returns the URL in a generic format. In the meantime, a subscription key is required for Bing Maps. The

Bing Maps key can be obtained from [here](#), then append it to the Bing Maps URL before passing it to the [getBingUrlTemplate](#) method. The URL returned by this method must be passed to the [urlTemplate](#) property.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' (load)="load($event)" style="display:block">
    <e-layers>
    <e-layer></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public load = (args: ILoadEventArgs) : void => {
    args.maps.getBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Metadata/Aerial?output=json&uriScheme=https&key=?").then(function(url) {
      args.maps.layers[0].urlTemplate= url;
    });
  };
}
```

Types of Bing Maps

Bing Maps provides different types of Maps and it is supported in the Maps component.

- **Aerial** - Displays satellite images to highlight roads and major landmarks for easy identification.
- **AerialWithLabel** - Displays aerial Maps with labels for the continent, country, ocean, etc.
- **Road** - Displays the default Maps view of roads, buildings, and geography.
- **CanvasDark** - Displays dark version of the road Maps.
- **CanvasLight** - Displays light version of the road Maps.
- **CanvasGray** - Displays grayscale version of the road Maps.

To render the light version of the road Maps, set the **CanvasLight** value is passed via the URL into the [getBingUrlTemplate](#) method demonstrated in the following code sample.

```
`typescript
```

```

import { Component, OnInit } from '@angular/core';
import { ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
@Component({
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' (load)="load($event)" style="display:block">
<e-layers>
<e-layer></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public load = (args: ILoadEventArgs) : void => {
    args.maps.getBingUrlTemplate("https://dev.virtualsearth.net/REST/V1/Imagery/Metadata/CanvasLight?
output=json&uriScheme=https&key=?").then(function(url) {
    args.maps.layers[0].urlTemplate= url;
  });
};
}
`

```

Enabling zooming and panning

Bing Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a Maps for in-depth analysis. Panning helps to move a Maps around to focus the targeted area.

```

`typescript
import { Component, OnInit, Inject } from '@angular/core';
import { Maps, Zoom, ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template: `
<ejs-maps id="rn-container" (load)="load($event)" [zoomSettings]="zoomSettings"
style="display:block">
<e-layers>
<e-layer></e-layer>

```

```

</e-layers>
</ejs-maps>
`

})
export class AppComponent implements OnInit {
  public zoomSettings: object;
  ngOnInit(): void {
    this.zoomSettings = {
      enable:true,
      toolbars: ["Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset"]
    };
    public load = (args: ILoadEventArgs) : void => {
      args.maps.getBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Metadata/Aerial?output=json&uriScheme=https&key=?").then(function(url) {
        args.maps.layers[0].urlTemplate= url;
      });
    };
  }
}
`

```

Adding markers and navigation line

Markers can be added to the layers of Bing Maps by setting the corresponding location's coordinates of latitude and longitude using [markerSettings](#). Navigation lines can be added on top of an Bing Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [navigationLineSettings](#).

```

`typescript
import { Component, OnInit, Inject } from '@angular/core';
import { Maps, Zoom, Marker, NavigationLine, ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom, Marker, NavigationLine);
@Component({
  selector: 'app-container',
  template: `
    <ejs-maps id="rn-container" (load)="load($event)" style="display:block" [zoomSettings]="zoomSettings"
    [centerPosition]="centerPosition">
    <e-layers>

```

```

<e-layer [markerSettings]="markerSettings" [navigationLineSettings]="navigationLineSettings"></e-
layer>
</e-layers>
</ejs-maps>
`
})
export class AppComponent implements OnInit {
  public zoomSettings: object;
  public centerPosition: object;
  public markerSettings: object;
  public navigationLineSettings: object;
  public load = (args: ILoadEventArgs) : void => {
    args.maps.getBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Metadata/Aerial?output
=json&uriScheme=https&key=?").then(function(url) {
    args.maps.layers[0].urlTemplate= url;
    });
  };
  ngOnInit(): void {
    this.zoomSettings = {
      zoomFactor: 4
    };
    this.centerPosition = {
      latitude: 29.394708,
      longitude: -94.954653
    };
    this.markerSettings = [
      {
        visible: true,
        height: 25,
        width: 15,
        dataSource: [
          {
            latitude: 34.06062,
            longitude: -118.330491,

```

```

name: 'California'
},
{
latitude: 40.724546,
longitude: -73.850344,
name: 'New York'
}
]
}
];
this.navigationLineSettings = [
{
visible: true,
color: 'blue',
width: 5,
angle: 0.1,
latitude: [34.06062, 40.724546],
longitude: [-118.330491, -73.850344]
}];
}
}
`

```

Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the Bing Maps layer for highlighting a particular continent or country in Bing Maps by adding another layer and specifying the [type](#) property of Maps layer to **SubLayer**.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { Maps, ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
import { africa_continent } from 'africa-continent.ts';
@Component({
selector: 'app-container',
template:
`<ejs-maps (load)="load($event)" id='rn-container' style="display:block">

```

```

<e-layers>
<e-layer></e-layer>
<e-layer [shapeData]= 'shapeData' [type] ='Sublayer' [shapeSettings]='shapeSettings'></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public load = (args: ILoadEventArgs) : void => {
    args.maps.getBingUrlTemplate("https://dev.virtualearth.net/REST/V1/Imagery/Metadata/Aerial?output=json&uriScheme=https&key=?").then(function(url) {
      args.maps.layers[0].urlTemplate= url;
    });
  };
  public shapeData: object;
  public shapeSettings: object;
  ngOnInit(): void {
    this.shapeData = africa_continent;
    this.shapeSettings = {
      fill: 'blue'
    }
  }
}
`

```

Enabling legend

The legend can be added to the tile Maps by setting the [visible](#) property of [legendSettings](#) to **true**.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { Maps, ILoadEventArgs } from '@syncfusion/ej2-angular-maps';
import { markerDataSource } from 'markerdata.ts';
@Component({
  selector: 'app-container',
  template:
    `<ejs-maps
    id="rn-container"

```

```

(load)="load($event)"
style="display:block"
[legendSettings]="legendSettings"

<e-layers>
<e-layer
[shapePropertyPath]="shapePropertyPath"
[shapeDataPath]="shapeDataPath"
[markerSettings]="markerSettings"
[shapeSettings]="shapeSettings"
</e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public load = (args: ILoadEventArgs): void => {
    args.maps
      .getBingUrlTemplate(
        'https://dev.virtualearth.net/REST/V1/Imagery/Metadata/Aerial?output=json&uriScheme=https&key=?'
      )
      .then(function (url) {
        args.maps.layers[0].urlTemplate = url;
      });
  };
  public shapeDataPath: string;
  public shapePropertyPath: string;
  public shapeSettings: object;
  public markerSettings: object;
  public legendSettings: object;
  ngOnInit(): void {
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.legendSettings = {

```



```

visible: true,
type: 'Markers',
useMarkerShape: true,
toggleLegendSettings: {
  enable: true,
  applyShapeSettings: false,
  border: {
    color: 'green',
    width: 2,
  },
},
};
this.shapeSettings = {
  fill: '#E5E5E5',
};
this.markerSettings = [
{
  dataSource: markerDataSource,
  colorValuePath: 'color',
  shapeValuePath: 'shape',
  legendText: 'country',
  visible: true,
},
];
}
}
,

```

Azure maps in Angular Maps component

Azure Maps is yet another online Maps provider, owned by Microsoft. As like OSM and Bing Maps, it provides Maps tile images based on our requests and combines those images into a single one to display Maps area.

Adding Azure Maps

The Azure Maps can be rendered using the [urlTemplate](#) property with the tile server URL provided by online map providers. In the meantime, a subscription key is required for Azure Maps. Follow the steps in this [link](#) to generate an API key, and then added the key to the URL.

Refer to [Azure Maps Licensing](#).

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

@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style="display:block">
    <e-layers>
    <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public urlTemplate: string;
  ngOnInit(): void {
    this.urlTemplate = 'https://atlas.microsoft.com/map/imagery/png?subscription-key=Your-Key &api-version=1.0&style=satellite&zoom=level&x=tileX&y=tileY';
  }
}
```

Enabling zooming and panning

The Azure Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a map for in-depth analysis. Panning helps to move a map around to focus the targeted area.

```
`typescript
import { Component, OnInit, Inject } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom);

@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style='display:block' [zoomSettings]='zoomSettings'>
    <e-layers>
    <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public zoomSettings: ZoomSettings;
  ngOnInit(): void {
    this.zoomSettings = {
      zoom: 10,
      pan: true
    };
  }
}
```

```

</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public zoomSettings: object;
  public urlTemplate: string;
  ngOnInit(): void {
    this.urlTemplate = 'https://atlas.microsoft.com/map/imagery/png?subscription-key=Your-Key &api-
    version=1.0&style=satellite&zoom=level&x=tileX&y=tileY';
    this.zoomSettings = {
      enable: true,
      toolbars: ["Zoom", "ZoomIn", "ZoomOut", "Pan", "Reset"]
    }
  }
}
`

```

Adding markers and navigation line

Markers can be added to the layers of Azure Maps by setting the corresponding location's coordinates of latitude and longitude using [markerSettings](#). Navigation lines can be added on top of the Azure Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [navigationLineSettings](#).

```

`typescript
import { Component, Inject, OnInit } from '@angular/core';
import { Maps, Zoom, Marker, NavigationLine } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom, Marker, NavigationLine);
@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style='display:block' [zoomSettings]='zoomSettings'
    [centerPosition]='centerPosition'>
    <e-layers>
    <e-layer [urlTemplate]='urlTemplate' [markerSettings]='markerSettings'
    [navigationLineSettings]='navigationLineSettings'></e-layer>
    </e-layers>
    </ejs-maps>`

```

```
})  
export class AppComponent implements OnInit {  
  public zoomSettings: object;  
  public centerPosition: object;  
  public markerSettings: object;  
  public navigationLineSettings: object;  
  public urlTemplate: string;  
  ngOnInit(): void {  
    this.urlTemplate = 'https://atlas.microsoft.com/map/imagery/png?subscription-key=Your-Key &api-version=1.0&style=satellite&zoom=level&x=tileX&y=tileY';  
    this.zoomSettings = {  
      zoomFactor: 4  
    };  
    this.centerPosition = {  
      latitude: 29.394708,  
      longitude: -94.954653  
    };  
    this.markerSettings = [{  
      visible: true,  
      height: 25,  
      width: 15,  
      dataSource: [  
        {  
          latitude: 34.060620,  
          longitude: -118.330491,  
          name: "California"  
        },  
        {  
          latitude: 40.724546,  
          longitude: -73.850344,  
          name: "New York"  
        }  
      ]  
    }  
  ]  
}
```

```

});
this.navigationLineSettings = [{
  visible: true,
  color: "blue",
  width: 5,
  angle: 0.1,
  latitude: [34.060620, 40.724546],
  longitude: [-118.330491,-73.850344]
}];
}
}
`

```

Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the Azure Maps layer for highlighting a particular continent or country in Azure Maps by adding another layer and specifying the [type](#) property of Maps layer to **SubLayer**.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { Maps } from '@syncfusion/ej2-angular-maps';
import { africa_continent } from 'africa-continent.ts';
@Component({
  selector: 'app-container',
  template:
    `

```

```

public urlTemplate: string;
ngOnInit(): void {
  this.urlTemplate = 'https://atlas.microsoft.com/map/imagery/png?subscription-key=Your-Key &api-version=1.0&style=satellite&zoom=level&x=tileX&y=tileY';
  this.type = 'SubLayer';
  this.shapeData = africa_continent;
  this.shapeSettings = {
    fill: 'blue'
  };
}

```

Enabling legend

The legend can be added to the Azure Maps by setting the [visible](#) property of [legendSettings](#) to **true**.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { Maps } from '@syncfusion/ej2-angular-maps';
import { markerDataSource } from 'markerdata.ts';

@Component({
  selector: 'app-container',
  template:
    `<ejs-maps
      id="rn-container"
      style="display:block"
      [legendSettings]="legendSettings"

      <e-layers>
      <e-layer [urlTemplate]= 'urlTemplate'
        [shapePropertyPath]="shapePropertyPath"
        [shapeDataPath]="shapeDataPath"
        [markerSettings]="markerSettings"
        [shapeSettings]="shapeSettings"
      </e-layer>
    </e-layers>

```

```
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeDataPath: string;
  public shapePropertyPath: string;
  public shapeSettings: object;
  public markerSettings: object;
  public legendSettings: object;
  public urlTemplate: string;
  ngOnInit(): void {
    this.urlTemplate = 'https://atlas.microsoft.com/map/imagery/png?subscription-key=Your-Key &api-version=1.0&style=satellite&zoom=level&x=tileX&y=tileY';
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.legendSettings = {
      visible: true,
      type: 'Markers',
      useMarkerShape: true,
      toggleLegendSettings: {
        enable: true,
        applyShapeSettings: false,
        border: {
          color: 'green',
          width: 2,
        },
      },
    };
    this.shapeSettings = {
      fill: '#E5E5E5',
    };
    this.markerSettings = [
      {
        dataSource: markerDataSource,
```

```

colorValuePath: 'color',
shapeValuePath: 'shape',
legendText: 'country',
visible: true,
},
];
}
}
,

```

Other maps in Angular Maps component

Apart from OpenStreetMap and Bing Maps, you can also render Maps from other online map service providers by specifying the URL provided by those providers in the [urlTemplate](#) property. The URL template concept has been implemented in such a way that any online map service providers using the following template can benefit from previewing their Map in the Syncfusion Angular Maps component.

<!-- markdownlint-disable MD034 -->

Sample Template: https://< domain_name >/maps/basic/{z}/{x}/{y}.png

- "{z}" - It represents zoom factor (level).
- "{x}" - It indicates tile image x-position (tileX).
- "{y}" - It indicates tile image y-position (tileY).

In this case, the key generated for those online map service providers can also be appended to the URL. This allows to create personalized Maps with your own content and imagery.

Following is an example of how to add a TomTom map. You can generate an API key by following the steps in this [link](#) and then adding the key to the URL.

```

`typescript
import { Component, OnInit } from '@angular/core';
import { Maps } from '@syncfusion/ej2-angular-maps';
@Component({
  selector: 'app-container',
  template:
    `

```



```

export class AppComponent implements OnInit {
  public urlTemplate: string;
  ngOnInit(): void {
    this.urlTemplate =
      "http://api.tomtom.com/map/1/tile/basic/main/level/tileX/tileY.png?key=subscription_key";
  }
}

```



Enabling zooming and panning

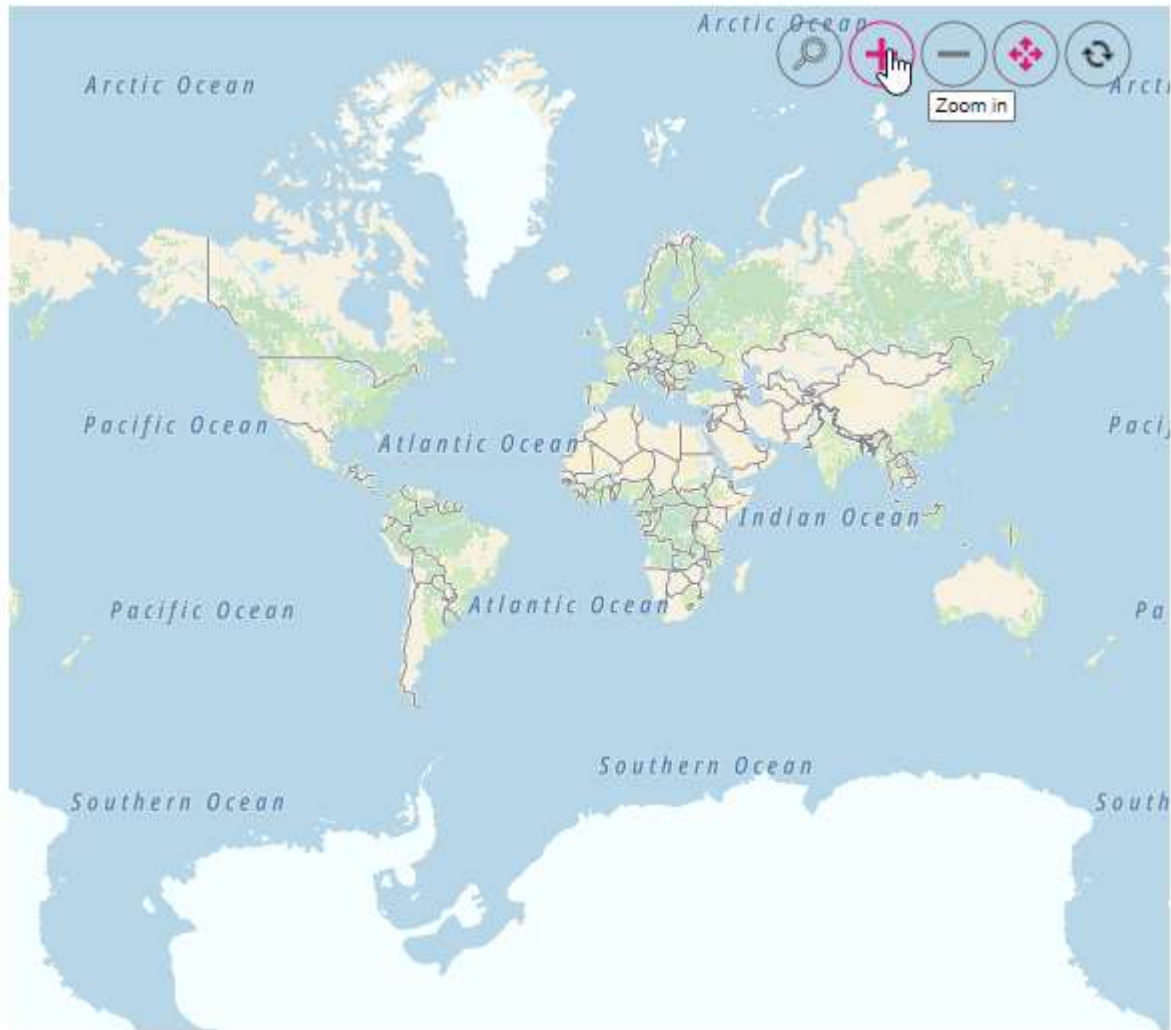
Tile Maps layer can be zoomed and panned. Zooming helps to get a closer look at a particular area on a map for in-depth analysis. Panning helps to move a map around to focus the targeted area.

```

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

```

```
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `
```



Adding markers and navigation line

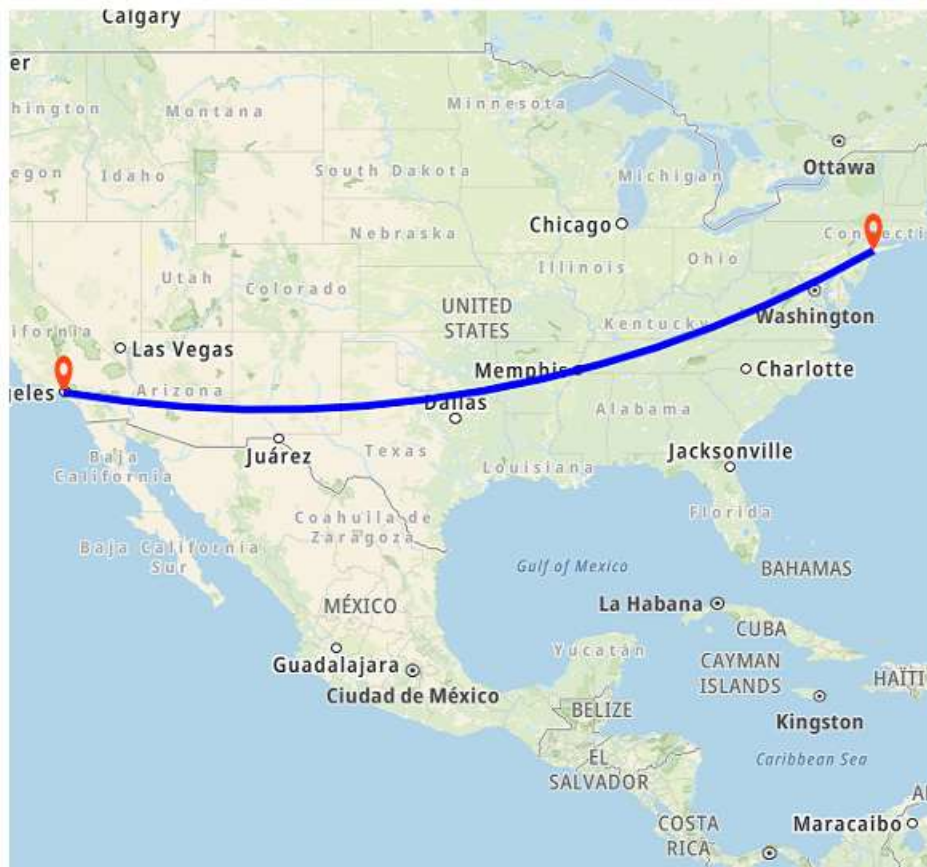
Markers can be added to the layers of tile Maps by setting the corresponding location's coordinates of latitude and longitude using [markerSettings](#) property. Navigation lines can be added on top of an tile Maps layer for highlighting a path among various places by setting the corresponding location's coordinates of latitude and longitude in the [navigationLineSettings](#).

``typescript`

```
import { Component, Inject, OnInit } from '@angular/core';
import { Maps, Zoom, Marker, NavigationLine } from '@syncfusion/ej2-angular-maps';
Maps.Inject(Zoom, Marker, NavigationLine);
@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' style="display:block" [zoomSettings]='zoomSettings'
    [centerPosition]='centerPosition'>
```

```
<e-layers>
<e-layer [markerSettings]='markerSettings' [navigationLineSettings]='navigationLineSettings'></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public urlTemplate: string;
  public zoomSettings: object;
  public markerSettings: object;
  public navigationLineSettings: object;
  ngOnInit(): void {
    this.urlTemplate =
      "http://api.tomtom.com/map/1/tile/basic/main/level/tileX/tileY.png?key=subscription_key";
    this.zoomSettings = {
      zoomFactor: 4
    };
    this.centerPosition = {
      latitude: 29.394708,
      longitude: -94.954653
    };
    this.markerSettings = [{
      visible: true,
      height: 25,
      width: 15,
      dataSource: [
        {
          latitude: 34.060620,
          longitude: -118.330491,
          name: "California"
        },
        {
          latitude: 40.724546,
          longitude: -73.850344,
```

```
name: "New York"  
}  
}  
});  
this.navigationLineSettings = [{  
  visible: true,  
  color: "blue",  
  width: 5,  
  angle: 0.1,  
  latitude: [34.060620, 40.724546],  
  longitude: [-118.330491, -73.850344]  
}];  
}  
}
```



Adding sublayer

Any GeoJSON shape can be rendered as a sublayer on top of the tile Maps layer for highlighting a particular continent or country in tile Maps by adding another layer and specifying the [type](#) property of Maps layer to **SubLayer**.

```
`typescript
```

```
import { Component, OnInit } from '@angular/core';
import { Maps } from '@syncfusion/ej2-angular-maps';
import { africa_continent } from 'africa-continent.ts';

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

```



Enabling legend

The legend can be added to the tile Maps by setting the [visible](#) property of [legendSettings](#) to **true**.

```
`typescript
import { Component, Inject, OnInit } from '@angular/core';
import { Maps } from '@syncfusion/ej2-angular-maps';
import { markerDataSource } from 'markerdata.ts';
@Component({
  selector: 'app-container',
  template:
`<ejs-maps
  id="rn-container"
  style="display:block"
  [legendSettings]="legendSettings"
```

```

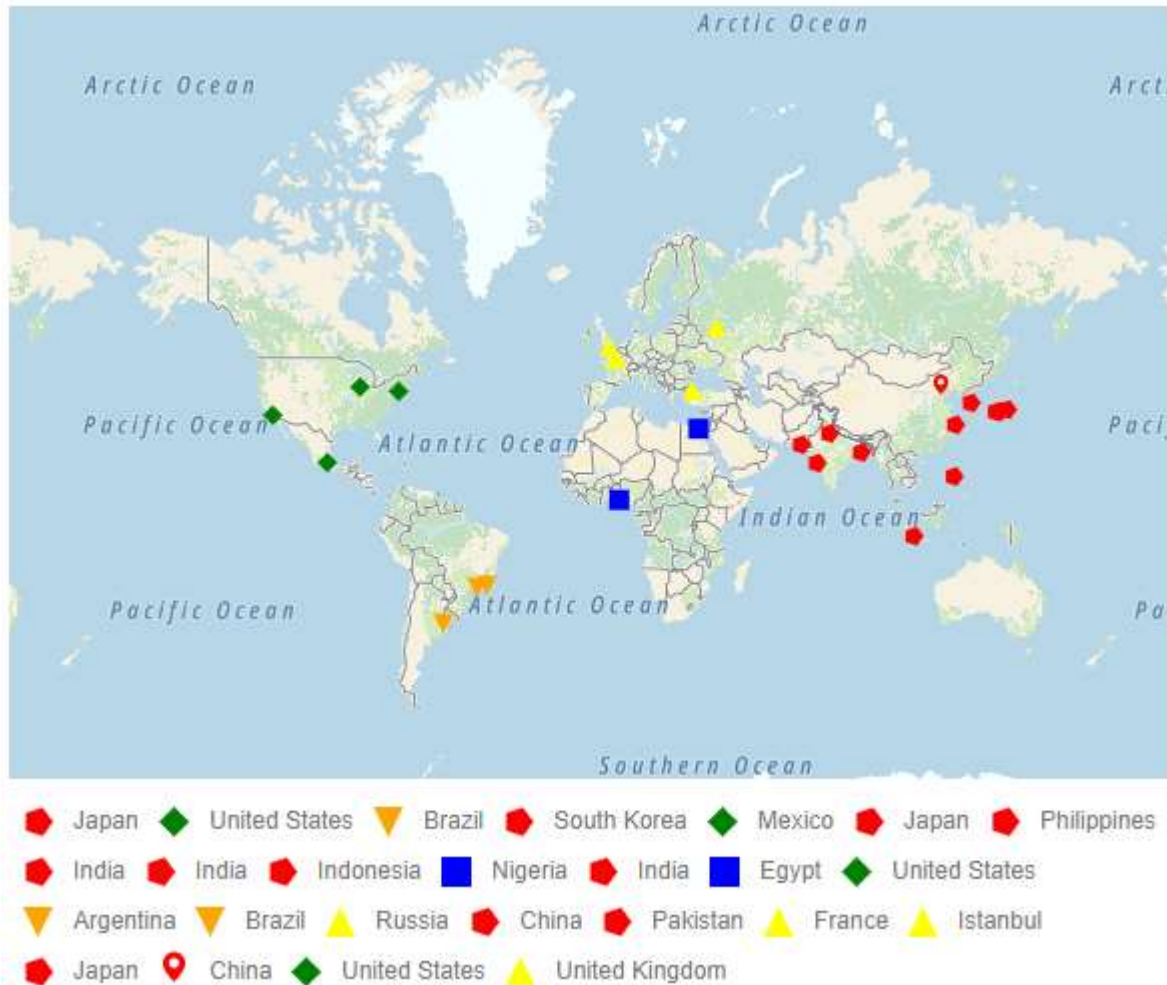
<e-layers>
  <e-layer [urlTemplate]= 'urlTemplate'
    [shapePropertyPath]="shapePropertyPath"
    [shapeDataPath]="shapeDataPath"
    [markerSettings]="markerSettings"
    [shapeSettings]="shapeSettings"
  </e-layer>
</e-layers>
</ejs-maps>`
})

export class AppComponent implements OnInit {
  public shapeDataPath: string;
  public shapePropertyPath: string;
  public shapeSettings: object;
  public markerSettings: object;
  public legendSettings: object;
  public urlTemplate: string;
  ngOnInit(): void {
    this.urlTemplate =
      'http://api.tomtom.com/map/1/tile/basic/main/level/tileX/tileY.png?key=subscription_key';
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.legendSettings = {
      visible: true,
      type: 'Markers',
      useMarkerShape: true,
      toggleLegendSettings: {
        enable: true,
        applyShapeSettings: false,
        border: {
          color: 'green',
          width: 2,

```



```
,  
,  
};  
this.shapeSettings = {  
  fill: '#E5E5E5',  
};  
this.markerSettings = [  
  {  
    dataSource: markerDataSource,  
    colorValuePath: 'color',  
    shapeValuePath: 'shape',  
    legendText: 'country',  
    visible: true,  
  },  
];  
}  
}  
,
```



Customization in Angular Maps component

Setting the size for Maps

The width and height of the Maps can be set using the [width](#) and [height](#) properties in the Maps component. Percentage or pixel values can be used for the height and width values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container' height='200px' width='200px'>
      <e-layers>
```

```

        <e-layer [shapeData]='shapeData'
[shapeSettings]="shapeSettings"></e-layer>
    </e-layers>
    </ejs-maps>`,
    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    public shapeData = world_map;
    public shapeSettings = {
      autofill: true
    }
  }
}

```

MAIN.TS

```

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

```

Maps title

The title for the Maps can be set using the [titleSettings](#). It can be customized using the following properties.

- [alignment](#) - To customize the alignment for the text in the title for the Maps. The possible values are **Center**, **Near** and **Far**.
- [description](#) - To set the description of the title in Maps.
- [text](#) - To set the text for the title in Maps.
- [textStyle](#) - To customize the text of the title in Maps.
- [subtitleSettings](#) - To customize the subtitle for the Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container' [titleSettings]='titleSettings'>
    <e-layers>
      <e-layer [shapeData]='shapeData'
[shapeSettings]="shapeSettings"></e-layer>
    </e-layers>
    </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})

```

```

    })
    export class AppComponent {
        public titleSettings: object = {
            text: 'Maps Control',
            textStyle: {
                color: 'red',
                fontStyle: 'italic',
                fontWeight: 'regular',
                fontFamily: 'arial',
                size: '14px'
            },
            alignment: 'Center'
        };
        public shapeData = world_map;
        public shapeSettings = {
            autofill: true
        }
    }
}

```

MAIN.TS

```

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

```

Setting theme

The Maps component supports following themes.

- Material
- Fabric
- Bootstrap
- Highcontrast
- MaterialDark
- FabricDark
- BootstrapDark
- Bootstrap4
- HighContrastLight
- Tailwind

By default, the Maps are rendered by the **Material** theme. The theme of the Maps component is changed using the [theme](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule

```

```

    ],
    standalone: true,
    selector: 'app-container',
    // specifies the template string for the maps component
    template:
      `

```

MAIN.TS

```

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

```

Customizing Maps container

The following properties are available to customize the container in the Maps.

- [background](#) - To apply the background color to the container in the Maps.
- [border](#) - To customize the color, width and opacity of the border of the Maps.
- [margin](#) - To customize the margins of the Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `

```

```

    </e-layers>
  </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public background: string = '#CCD1D1';
  public border: object = {
    color: 'green',
    width: 2
  };
  public margin: object = {
    bottom: 10,
    left: 20,
    right: 20,
    top: 10
  }
  public shapeData = world_map;
  public shapeSettings = {
    autofill: true
  }
}
}

```

MAIN.TS

```

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

```

Customizing Maps area

By default, the background color of the shape maps is set as **white**. To modify the background color of the Maps area, the [background](#) property in the [mapsArea](#) is used. The border of the Maps area can be customized using [border](#) property in the [mapsArea](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container' [mapsArea]='mapsArea'>
      <e-layers>
        <e-layer [shapeData]='shapeData'
[shapeSettings]="shapeSettings"></e-layer>
      </e-layers>
    </ejs-maps>`,

```

```

    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    public mapsArea: object = {
      background: '#CCD1D1',
      border: {
        width: 2,
        color: 'green'
      }
    };
    public shapeData = world_map;
    public shapeSettings = {
      autofill: true
    }
  }
}

```

MAIN.TS

```

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

```

Customizing the shapes

The following properties are available in [shapeSettings](#) to customize the shapes of the Maps.

- [fill](#) - To apply the fill color to the all the shapes.
- [autofill](#) - To apply the palette colors to the shapes if it is set as true.
- [palette](#) - To set the custom palette for the shapes.
- [border](#) - To customize the color, width and opacity of the border of the shapes.
- [dashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the shapes.
- [opacity](#) - To customize the transparency for the shapes.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]='shapeData' [shapeSettings]="shapeSettings"></e-
layer>
      </e-layers>
    </ejs-maps>`,

```

```

    encapsulation: ViewEncapsulation.None
  })
  export class AppComponent {
    public shapeData = world_map;
    public shapeSettings = {
      autofill: true,
      palette: ['#C7DE6C', '#59A076', '#88D0BC', '#FEA78C', '#FFC557'],
      border: {
        color: '#FEE1DD',
        width: 3
      },
      dashArray: '1',
      opacity: 0.9
    };
  }
}

```

MAIN.TS

```

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

```

Setting color to the shapes from the data source

The color for each shape in the Maps can be set using the [colorValuePath](#) property of [shapeSettings](#). The value for the [colorValuePath](#) property is the field name from the data source of the [shapeSettings](#) which contains the color values.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]='shapeData' [shapePropertyPath]='shapePropertyPath'
        [shapeDataPath]='shapeDataPath' [dataSource]='dataSource'
        [shapeSettings]="shapeSettings"></e-layer>
      </e-layers>
    </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public shapeData = world_map;
  public shapePropertyPath = 'continent';
}

```



```

public shapeDataPath = 'continent';
public dataSource = [
  { continent: "North America", color: '#71B081' },
  { continent: "South America", color: '#5A9A77' },
  { continent: "Africa", color: '#498770' },
  { continent: "Europe", color: '#39776C' },
  { continent: "Asia", color: '#266665' },
  { continent: "Oceania", color: '#124F5E' }
]
public shapeSettings = {
  colorValuePath: 'color'
};
}

```

MAIN.TS

```

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

```

Applying border to individual shapes

The border of each shape in the Maps can be customized using the [borderColorValuePath](#) and [borderWidthValuePath](#) properties to modify the color and the width of the border respectively. The field name in the data source of the layer which contains the color and the width values must be set in the [borderColorValuePath](#) and [borderWidthValuePath](#) properties respectively. If the values of [borderWidthValuePath](#) and [borderColorValuePath](#) do not match with the field name from the data source, then the color and width of the border will be applied to the shapes using the [border](#) property in the [shapeSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]='shapeData' [shapePropertyPath]='shapePropertyPath'
        [shapeDataPath]='shapeDataPath' [dataSource]='dataSource'
        [shapeSettings]="shapeSettings"></e-layer>
      </e-layers>
    </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})

```

```
export class AppComponent {
  public shapeData = world_map;
  public shapePropertyPath = 'continent';
  public shapeDataPath = 'continent';
  public dataSource = [
    { continent: "North America", color: '#71B081', borderColor:
'#CCFFE5', width: 2 },
    { continent: "South America", color: '#5A9A77', borderColor: 'red',
width: 2 },
    { continent: "Africa", color: '#498770', borderColor: '#FFCC99',
width: 2 },
    { continent: "Europe", color: '#39776C', borderColor: '#66B2FF',
width: 2 },
    { continent: "Asia", color: '#266665', borderColor: '#999900',
width: 2 },
    { continent: "Oceania", color: '#124F5E', borderColor: 'blue',
width: 2 }
  ]
  public shapeSettings = {
    borderColorValuePath: 'borderColor',
    borderWidthValuePath: 'width',
    colorValuePath: 'color'
  };
}
```

MAIN.TS

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

Projection type

The Maps component supports the following projection types:

- Mercator
- Equiarectangular
- Miller
- Eckert3
- Eckert5
- Eckert6
- Winkel3
- AitOff

By default, the Maps are rendered by the **Mercator** projection type in which the Maps are rendered based on the coordinates. So, the Maps is not stretched. To change the type of projection in the Maps, the [projectionType](#) property is used.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
```

```
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template:
    `<div>
      <div>
        <ejs-maps id='maps' #maps style="display:block;"
projectionType="Miller">
          <e-layers>
            <e-layer [shapeData]='shapeData'></e-layer>
          </e-layers>
        </ejs-maps>
      </div>
    </div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public shapeData = world_map;
}
```

MAIN.TS

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

Color mapping in Angular Maps component

Color mapping is used to customize the shape colors based on the given values. It has three types.

1. Range color mapping
2. Equal color mapping
3. Desaturation color mapping.

To add color mapping to the shapes of the Maps, bind the data source to the [dataSource](#) property of [layerSettings](#) and set the field name which contains the color value in the data source to the [colorValuePath](#) property.

Types of color mapping

Range color mapping

Range color mapping applies the color to the shapes of the Maps which matches the numeric values in the data source within the given color mapping ranges. The [from](#) and [to](#) properties in the [colorMapping](#) are used to specify the color mapping ranges in the Maps.

`typescript

```
export let population_density = [
```

<https://ej2.syncfusion.com/angular/documentation>.

```
{
  'code': 'AE',
  'value': 90,
  'name': 'United Arab Emirates',
  'population': 8264070,
  'density': 99
},
{
  'code': 'GB',
  'value': 257,
  'name': 'United Kingdom',
  'population': 62041708,
  'density': 255
},
{
  'code': 'US',
  'value': 34,
  'name': 'United States',
  'population': 325020000,
  'density': 33
}
...
];
`
```

Bind the **population_density** data to the [dataSource](#) property of [layerSettings](#) and set the [colorValuePath](#) property of [shapeSettings](#) as **density**. The range values can be set using the [from](#) and [to](#) properties of [colorMapping](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from '../world-map';
import { Population_Density } from '../data';
@Component({
  imports: [
```

```

    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeDataPath] = 'shapeDataPath'
        [shapePropertyPath] ='shapePropertyPath' [shapeSettings] = 'shapeSettings'
        [dataSource] ='dataSource'></e-layer>
      </e-layers>
    </ejs-maps>`
  })
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public dataSource?: object[];
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath= 'name';
    this.dataSource= Population_Density;
    this.shapeSettings = {
      colorValuePath: 'density',
      fill: '#E5E5E5',
      colorMapping: [
        {
          from: 0.00001, to: 100, color: 'rgb(153,174,214)'
        },
        {
          from: 100, to: 200, color: 'rgb(115,143,199)'
        },
        {
          from: 200, to: 300, color: 'rgb(77,112,184)'
        },
        {
          from: 300, to: 500, color: 'rgb(38,82,168)'
        },
        {
          from: 500, to: 19000, color: 'rgb(0,51,153)'
        }
      ]
    };
  }
}

```

MAIN.TS

```

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

```

Equal color mapping

Equal color mapping applies the color to the shapes of the Maps when the [value](#) property of [colorMapping](#) matches with the values provided in the data source. The following example shows how to apply equal color mapping to the shapes with the data source **unCountries** which illustrates the permanent and non-permanent countries in the UN security council.

```
`typescript
```

```
export let unCountries: object[] = [
  { Country: 'China', Membership: 'Permanent' },
  { Country: 'France', Membership: 'Permanent' },
  { Country: 'Russia', Membership: 'Permanent' },
  { Country: 'United Kingdom', Membership: 'Permanent' },
  { Country: 'United States', Membership: 'Permanent' },
  { Country: 'Bolivia', Membership: 'Non-Permanent' },
  { Country: 'Eq. Guinea', Membership: 'Non-Permanent' },
  { Country: 'Ethiopia', Membership: 'Non-Permanent' },
  { Country: "Côte d'Ivoire", Membership: 'Permanent' },
  { Country: 'Kazakhstan', Membership: 'Non-Permanent' },
  { Country: 'Kuwait', Membership: 'Non-Permanent' },
  { Country: 'Netherlands', Membership: 'Non-Permanent' },
  { Country: 'Peru', Membership: 'Non-Permanent' },
  { Country: 'Poland', Membership: 'Non-Permanent' },
  { Country: 'Sweden', Membership: 'Non-Permanent' },
];
```

Bind the **unCountries** data to the [dataSource](#) property of [layerSettings](#) and set the [colorValuePath](#) property of [shapeSettings](#) as **Membership**. Set the [value](#) property in the [colorMapping](#) property to **Permanent** and **Non-Permanent** in the different set of color mapping properties. If the corresponding value of the [colorValuePath](#) property matches with the corresponding field name in the data source, then the given color will be applied.

```
[app.component.ts]
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { unCountries } from './data';
@Component({
  imports: [
```

```

    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeDataPath] = 'shapeDataPath'
        [shapePropertyPath] = 'shapePropertyPath' [shapeSettings] = 'shapeSettings'
        [dataSource] = 'dataSource'></e-layer>
      </e-layers>
    </ejs-maps>`
  })
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public dataSource?: object[];
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'Country';
    this.shapePropertyPath = 'name';
    this.dataSource = unCountries;
    this.shapeSettings = {
      fill: '#E5E5E5',
      colorMapping: [
        {
          value: 'Permanent',
          color: '#C3E6ED'
        },
        {
          color: '#F1931B',
          value: 'Non-Permanent'
        }
      ],
      colorValuePath: 'Membership'
    }
  }
}

```

MAIN.TS

```

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

```

Desaturation color mapping

Desaturation color mapping applies the color to the shapes of the Maps, similar to the range color mapping. The opacity will be applied in this color mapping based on the [minOpacity](#) and [maxOpacity](#) properties in the [colorMapping](#).

The following example shows how to apply desaturation color mapping to the shapes with the data source **population_density** that is available in the [Range color mapping](#) section.

Bind the **population_density** data to the [dataSource](#) property of [layerSettings](#) and set the [colorValuePath](#) property of [shapeSettings](#) as **density**. The range values can be set using the [from](#) and [to](#) properties of [colorMapping](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { Population_Density } from './data';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeDataPath] = 'shapeDataPath'
[shapePropertyPath] ='shapePropertyPath' [shapeSettings] = 'shapeSettings'
[dataSource] ='dataSource'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public dataSource?: object[];
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath= 'name';
    this.dataSource= Population_Density;
    this.shapeSettings = {
      colorValuePath: 'density',
      fill: '#E5E5E5',
      colorMapping: [
        {
          from: 0, to: 100, minOpacity: 0.4, maxOpacity: 1, color:
'rgb(153,174,214)'
        },
        {
          from: 101, to: 200, minOpacity: 0.7, maxOpacity: 1,
color: 'rgb(115,143,199)'
        }
      ]
    };
  }
}
```

MAIN.TS


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

Multiple colors for a single shape

Multiple colors can be added to the color mapping which can be used as gradient effect to a specific shape based on the ranges in the data source. By using the [color](#) property of [colorMapping](#), any number of colors can be set to the shapes as a gradient.

The following example demonstrates how to use multiple colors in color mapping with the data source **population_density** that is available in the [Range color mapping](#) section.

Bind the **population_density** data to the [dataSource](#) property of [layerSettings](#) and set the [colorValuePath](#) property of [shapeSettings](#) as **density**. The range values can be set using the [from](#) and [to](#) properties of [colorMapping](#).

[app.component.ts]

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { Population_Density } from './data';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeDataPath] = 'shapeDataPath'
    [shapePropertyPath] = 'shapePropertyPath' [shapeSettings] = 'shapeSettings'
    [dataSource] = 'dataSource'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public dataSource?: object[];
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.dataSource = Population_Density;
    this.shapeSettings = {
      colorValuePath: 'density',
      fill: '#E5E5E5',
```

```

        colorMapping: [
            {
                from: 0, to: 100, color: ['red', 'blue']
            },
            {
                from: 101, to: 200, color: ['green', 'violet']
            },
        ]
    };
}
}

```

MAIN.TS

```

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

```

Color for items excluded from color mapping

Color mapping can be applied to the shapes in the Maps which does not match color mapping criteria such as range or equal values using the [color](#) property of [colorMapping](#).

The following example shows how to set the color for items excluded from the color mapping with the data source **population_density** that is available in the [Range color mapping](#) section.

In the following example, color mapping is added for the ranges from 0 to 200. If there are any records in the data source that are outside of this range, the color mapping will not be applied. To apply the color for these excluded items, set the [color](#) property alone in the [colorMapping](#).

[app.component.ts]

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { Population_Density } from './data';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeDataPath] = 'shapeDataPath'
    [shapePropertyPath] = 'shapePropertyPath' [shapeSettings] = 'shapeSettings'
    [dataSource] = 'dataSource'></e-layer>
    </e-layers>
    </ejs-maps>`
})

```

```
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public dataSource?: object[];
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.dataSource = Population_Density;
    this.shapeSettings = {
      colorValuePath: 'density',
      fill: '#E5E5E5',
      colorMapping: [
        {
          from: 0, to: 100, color: 'red'
        },
        {
          from: 101, to: 200, color: 'green'
        },
        {
          color: 'violet'
        }
      ]
    };
  }
}
```

MAIN.TS

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

Color mapping for bubbles

The color mapping types such as range color mapping, equal color mapping and desaturation color mapping are applicable for bubbles in the Maps. To add color mapping for bubbles of the Maps, bind the data source to the [dataSource](#) property of [bubbleSettings](#) and set the field name which contains the color value in the data source to the [colorValuePath](#) property. Multiple colors for a single set of bubbles and color for excluded items from [colorMapping](#) are also applicable for bubbles.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { BubbleService } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
```

```

providers: [BubbleService],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeDataPath]='shapeDataPath'
[shapePropertyPath]='shapePropertyPath' [bubbleSettings] =
'bubbleSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
  })
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 5,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'New Zealand', population: '19651127' },
        { name: 'Pakistan', population: '3090416' }
      ],
      valuePath: 'population',
      colorValuePath: 'population',
      colorMapping: [{
        value: '38332521',
        color: '#C3E6ED'
      },
      {
        value: '19651127',
        color: '#F1931B'
      },
      {
        value: '3090416',
        color: 'blue'
      }
    ]
  }]
}
}

```

MAIN.TS

```

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

```

Data labels in Angular Maps component

Data labels provide information to users about the shapes of the Maps component. It can be enabled by setting the [visible](#) property of the [dataLabelSettings](#) to **true**.

Adding data labels

To display data labels in the Maps, the [labelPath](#) property of [dataLabelSettings](#) must be used. The value of the [labelPath](#) property can be taken from the field name in the shape data or data source. In the following example, the value of the [labelPath](#) property is the field name in the shape data of the Maps layer.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [DataLabelService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
    [dataLabelSettings] = 'dataLabelSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object
  ngOnInit(): void {
    this.shapeData = usa_map;
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {
      visible: true,
      labelPath: 'name',
    };
  }
}
```

MAIN.TS

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

In the following example, the value of [labelPath](#) property is set from the field name in the data source of the layer settings.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [DataLabelService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData'
    [shapePropertyPath]='shapePropertyPath' [shapeDataPath]='shapeDataPath'
    [shapeSettings] = 'shapeSettings' [dataSource]='dataSource'
    [dataLabelSettings] = 'dataLabelSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public dataSource?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'name';
    this.dataSource = [
      {"name": "Afghanistan", "value": 53, "countryCode": "AF",
      "population": "29863010", "color": "red", "density": "119", "continent":
      "Asia"},
      {"name": "Albania", "value": 117, "countryCode": "AL",
      "population": "3195000", "color": "Blue", "density": "111", "continent":
      "Europe"},
      {"name": "Algeria", "value": 15, "countryCode": "DZ",
      "population": "34895000", "color": "Green", "density": "15", "continent":
      "Africa"}]
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {
      visible: true,
      labelPath: "continent",

```

```

        smartLabelMode: 'Trim'
    };
}
}

```

MAIN.TS

```

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

```

Customization

The following properties are available in the [dataLabelSettings](#) to customize the data label of the Maps component.

- [border](#) - To customize the color, width and opacity for the border of the data labels in Maps.
- [fill](#) - To apply the color of the data labels in Maps.
- [opacity](#) - To customize the transparency of the data labels in Maps.
- [textStyle](#) - To customize the text style of the data labels in Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [DataLabelService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
    [dataLabelSettings] = 'dataLabelSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = usa_map;
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {

```

```

        visible: true,
        labelPath: 'name',
        border: {
            color: 'green',
            width: 2
        },
        fill: 'red',
        opacity: 0.9,
        textStyle: {
            color: 'blue',
            size: '10px',
            fontStyle: 'Sans-serif',
            fontWeight: 'normal',
        }
    };
}
}

```

MAIN.TS

```

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

```

Label animation

The data labels can be animated during the initial rendering of the Maps. This can be enabled by setting the [animationDuration](#) property in the `dataLabelSettings` of the Maps. The duration of the animation is specified in milliseconds.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService, DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService, DataLabelService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
        [dataLabelSettings] = 'dataLabelSettings' [tooltipSettings]
        ='tooltipSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
})

```



```
export class AppComponent implements OnInit {
  public tooltipSettings?: object;
  public shapeData?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = usa_map;
    this.shapeSettings = {
      autofill: true
    };
    this.tooltipSettings = {
      visible: true,
      valuePath: 'name'
    };
    this.dataLabelSettings = {
      visible: true,
      smartLabelMode: 'Hide',
      intersectionAction: 'Trim',
      labelPath: 'name',
      animationDuration: 2000
    };
  }
}
```

MAIN.TS

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

Smart labels

The Maps component provides an option to handle the labels when they intersect with the corresponding shape borders using the [smartLabelMode](#) property. The following options are available in the [smartLabelMode](#) property.

- None
- Hide
- Trim

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [DataLabelService],
  standalone: true,
```

```

    selector: 'app-container',
    template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
    [dataLabelSettings] = 'dataLabelSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public shapeData?: object;
    public shapeSettings?: object;
    public dataLabelSettings?: object;
    ngOnInit(): void {
      this.shapeData = usa_map;
      this.shapeSettings = {
        autofill: true
      };
      this.dataLabelSettings = {
        visible: true,
        labelPath: 'name',
        smartLabelMode: 'Hide'
      };
    }
  }
}

```

MAIN.TS

```

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

```

Intersect action

The Maps component provides an option to handle the labels when a label intersects with another label using the [intersectionAction](#) property. The following options are available in the [intersectionAction](#) property.

- None
- Hide
- Trim

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],

```

```

providers: [DataLabelService],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
[dataLabelSettings] = 'dataLabelSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
}))
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeSettings?: object;
  public dataLabelSettings?: object;
  ngOnInit(): void {
    this.shapeData = usa_map;
    this.shapeSettings = {
      autofill: true
    };
    this.dataLabelSettings = {
      visible: true,
      labelPath: 'name',
      intersectionAction: 'Trim'
    };
  }
}

```

MAIN.TS

```

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

```

Adding data label as a template

The data label can be added as a template in the Maps component. The [template](#) property of [dataLabelSettings](#) is used to set the data label as a template. Any text or HTML element can be added as the template in data labels.

The properties of data label such as, `smartLabelMode`, `intersectionAction`, `animationDuration`, `border`, `fill`, `opacity` and `textStyle` properties are not applicable to `template` property. The styles can be applied to the label template using the CSS styles of the HTML element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { DataLabelService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule

```

```

    ],
    providers: [DataLabelService],
    standalone: true,
    selector: 'app-container',
    template:
      `<ejs-maps id='rn-container'>
        <e-layers>
          <e-layer [shapeData]= 'shapeData' [shapeSettings] = 'shapeSettings'
            [dataLabelSettings] = 'dataLabelSettings'></e-layer>
        </e-layers>
      </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public shapeData?: object;
    public shapeSettings?: object;
    public dataLabelSettings?: object;
    ngOnInit(): void {
      this.shapeData = usa_map;
      this.shapeSettings = {
        autofill: true
      };
      this.dataLabelSettings = {
        visible: true,
        labelPath: 'name',
        template: 'Label'
      };
    }
  }
}

```

MAIN.TS

```

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

```

Polygon shape in Angular Maps component

The Maps component allows you to add polygon shape to a geometry map or an online map by using the properties in the [polygons](#). This section describes how to add polygon shape to the map and customize them.

Adding polygon shape

To render polygon shape in Maps, **Polygon** module must be injected into the Maps using **Maps.Inject(Polygon)** method. The polygon shape can be rendered over the map layer by defining the [points](#) property in the **polygons** of the Maps component. The **points** property uses a collection of latitude and longitude values to define the polygon shape.

The **polygons** provides the following properties for customizing the polygon shape of the Maps component.

- [fill](#) - It is used to change the color of the polygon shape.
- [opacity](#) - It is used to change the opacity of the polygon shape.
- [borderColor](#) - It is used to change the color of the border in the polygon shape.

- [borderWidth](#) - It is used to change the width of the border in the polygon shape.
- [borderOpacity](#) - It is used to change the opacity of the border in the polygon shape.

You can also include “n” polygon shapes using the [polygons](#) property.

The following example shows how to customize the polygon shape over the geometry map.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule, PolygonService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PolygonService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="control-section">
<div align='center'>
<ejs-maps id='container' style="display:block;">
<e-layers>
<e-layer [shapeData]='shapeData' [polygonSettings]='polygonSettings'></e-
layer>
</e-layers>
</ejs-maps>
</div>
</div>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public polygonSettings?: object;
  ngOnInit(): void {
    this.polygonSettings = {
      polygons: [
        {
          points: [
            { longitude: 34.88539587371454, latitude: 28.181421087099537 },
            { longitude: 37.50029619722466, latitude: 24.299419888989462 },
            { longitude: 39.22241423764024, latitude: 22.638529461838658 },
            { longitude: 38.95650769309776, latitude: 21.424998160017495 },
            { longitude: 40.19963938650778, latitude: 20.271205391339606 },
            { longitude: 41.76547269134551, latitude: 18.315451049867193 },
            { longitude: 42.78452077838921, latitude: 16.097235052947966 },
            { longitude: 43.36984949591576, latitude: 17.188572054533054 },
            { longitude: 44.12558191797012, latitude: 17.407258102232234 },
            { longitude: 46.69027032797584, latitude: 17.33342243475734 },
            { longitude: 47.09312386141585, latitude: 16.97087769526452 },
            { longitude: 48.3417299826302, latitude: 18.152700711188004 },
            { longitude: 49.74762591400318, latitude: 18.81544363931681 },
            { longitude: 52.41428026336621, latitude: 18.9035706497573 },
            { longitude: 55.272683129240335, latitude: 20.133861012918544 },
            { longitude: 55.60121336079203, latitude: 21.92042703112351 },
          ]
        }
      ]
    }
  }
}
```

```

    { longitude: 55.08204399107967, latitude: 22.82330266225882 },
    { longitude: 52.743894337844154, latitude: 22.954463486477437 },
    { longitude: 51.47035908651375, latitude: 24.35818837668566 },
    { longitude: 51.122553219055874, latitude: 24.666679732426346 },
    { longitude: 51.58731671256831, latitude: 25.173806925822717 },
    { longitude: 51.35950585992913, latitude: 25.84556484481108 },
    { longitude: 51.088770529661275, latitude: 26.168494193631147 },
    { longitude: 50.78527056538036, latitude: 25.349051242147596 },
    { longitude: 50.88330288802666, latitude: 24.779242606720743 },
    { longitude: 50.19702490702369, latitude: 25.66825106363693 },
    { longitude: 50.066461167339924, latitude: 26.268905608606616 },
    { longitude: 49.645896067213215, latitude: 27.15116474192905 },
    { longitude: 48.917371072320805, latitude: 27.55738830340198 },
    { longitude: 48.3984720209192, latitude: 28.566207269716173 },
    { longitude: 47.68851714518985, latitude: 28.5938991332588 },
    { longitude: 47.45059089191449, latitude: 29.009321449856984 },
    { longitude: 44.73549453609391, latitude: 29.157358362696385 },
    { longitude: 41.79487372890989, latitude: 31.23489959729713 },
    { longitude: 40.36977176033773, latitude: 31.9642352513131 },
    { longitude: 39.168270913149826, latitude: 32.18348471414393 },
    { longitude: 37.019253492546454, latitude: 31.47710220862595 },
    { longitude: 37.99644645508337, latitude: 30.4851028633376 },
    { longitude: 37.67756530485232, latitude: 30.3636358598429 },
    { longitude: 37.50181466030105, latitude: 29.960155516804974 },
    { longitude: 36.700288181129594, latitude: 29.882136586478993 },
    { longitude: 36.100009274845206, latitude: 29.15308642012721 },
    { longitude: 34.85774476486728, latitude: 29.3103032832622 },
    { longitude: 34.64498583263142, latitude: 28.135787235699823 },
    { longitude: 34.88539587371454, latitude: 28.181421087099537 }
  ],
  fill: 'red',
  opacity: 0.7,
  borderColor: 'green',
  borderWidth: 2,
  borderOpacity: 0.7
}
];
this.shapeData = world_map;
}
}

```

MAIN.TS

```

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

```

Tooltip

Tooltip is used to display more information about a polygon shape during a mouse or touch interaction. Tooltip and tooltip template can be enabled by setting the [visible](#) property to **true** in the [tooltipSettings](#). Additionally, you need to set the desired content as a value to the [tooltipText](#) property in the `polygons`

property to show the tooltip. If you add 'n' numbers of polygon shapes, you can add the `tooltipText` property to each polygon, which will display the tooltip for the associated polygon shape.

Tooltip customization

The following properties are available in the `tooltipSettings` to customize the appearance of the tooltip.

- `border` - To change the color, width, and opacity of the border of the tooltip in the polygon shape.
- `fill` - Applies the color of the tooltip in the polygon shape.
- `textStyle` - To change the style of the text in the tooltip of the polygon shape.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule, PolygonService, MapsTooltipService } from
 '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PolygonService, MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="control-section">
    <div align="center">
      <ejs-maps id="container" style="display:block;">
        <e-layers>
          <e-layer [shapeData]='shapeData' [polygonSettings]='polygonSettings'></e-
layer>
        </e-layers>
      </ejs-maps>
    </div>
  </div>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public polygonSettings?: object;
  ngOnInit(): void {
    this.polygonSettings = {
      tooltipSettings: { visible: true, border: { width: 2, color: 'red' } }
    },
    polygons: [
      {
        tooltipText: 'Saudi Arabia',
        points: [
          { longitude: 34.88539587371454, latitude: 28.181421087099537 },
          { longitude: 37.50029619722466, latitude: 24.299419888989462 },
          { longitude: 39.22241423764024, latitude: 22.638529461838658 },
          { longitude: 38.95650769309776, latitude: 21.424998160017495 },
          { longitude: 40.19963938650778, latitude: 20.271205391339606 },
          { longitude: 41.76547269134551, latitude: 18.315451049867193 },
        ]
      }
    ]
  }
}
```

```

    { longitude: 42.78452077838921, latitude: 16.097235052947966 },
    { longitude: 43.36984949591576, latitude: 17.188572054533054 },
    { longitude: 44.12558191797012, latitude: 17.407258102232234 },
    { longitude: 46.69027032797584, latitude: 17.33342243475734 },
    { longitude: 47.09312386141585, latitude: 16.97087769526452 },
    { longitude: 48.3417299826302, latitude: 18.152700711188004 },
    { longitude: 49.74762591400318, latitude: 18.81544363931681 },
    { longitude: 52.41428026336621, latitude: 18.9035706497573 },
    { longitude: 55.272683129240335, latitude: 20.133861012918544 },
    { longitude: 55.60121336079203, latitude: 21.92042703112351 },
    { longitude: 55.08204399107967, latitude: 22.823302662258882 },
    { longitude: 52.743894337844154, latitude: 22.954463486477437 },
    { longitude: 51.47035908651375, latitude: 24.35818837668566 },
    { longitude: 51.122553219055874, latitude: 24.666679732426346 },
    { longitude: 51.58731671256831, latitude: 25.173806925822717 },
    { longitude: 51.35950585992913, latitude: 25.84556484481108 },
    { longitude: 51.088770529661275, latitude: 26.168494193631147 },
    { longitude: 50.78527056538036, latitude: 25.349051242147596 },
    { longitude: 50.88330288802666, latitude: 24.779242606720743 },
    { longitude: 50.19702490702369, latitude: 25.66825106363693 },
    { longitude: 50.066461167339924, latitude: 26.268905608606616 },
    { longitude: 49.645896067213215, latitude: 27.15116474192905 },
    { longitude: 48.917371072320805, latitude: 27.55738830340198 },
    { longitude: 48.3984720209192, latitude: 28.566207269716173 },
    { longitude: 47.68851714518985, latitude: 28.5938991332588 },
    { longitude: 47.45059089191449, latitude: 29.009321449856984 },
    { longitude: 44.73549453609391, latitude: 29.157358362696385 },
    { longitude: 41.79487372890989, latitude: 31.23489959729713 },
    { longitude: 40.36977176033773, latitude: 31.9642352513131 },
    { longitude: 39.168270913149826, latitude: 32.18348471414393 },
    { longitude: 37.019253492546454, latitude: 31.47710220862595 },
    { longitude: 37.99644645508337, latitude: 30.4851028633376 },
    { longitude: 37.67756530485232, latitude: 30.3636358598429 },
    { longitude: 37.50181466030105, latitude: 29.960155516804974 },
    { longitude: 36.700288181129594, latitude: 29.882136586478993 },
    { longitude: 36.100009274845206, latitude: 29.15308642012721 },
    { longitude: 34.85774476486728, latitude: 29.3103032832622 },
    { longitude: 34.64498583263142, latitude: 28.135787235699823 },
    { longitude: 34.88539587371454, latitude: 28.181421087099537 }
  ],
  fill: 'red',
  opacity: 0.7,
  borderColor: 'green',
  borderWidth: 2,
  borderOpacity: 0.7
}
]
};
this.shapeData = world_map;
}
}

```

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

Any HTML element can be rendered in the tooltip of the polygon shapes using the [tooltipTemplate](#) property of the `polygons` property. If you add 'n' numbers of polygon shapes, you can add the `tooltipTemplate` property to each polygon, which will display the tooltip for the associated polygon shape.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule, PolygonService, MapsTooltipService } from
 '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PolygonService, MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="control-section">
    <div align="center">
      <ejs-maps id="container" style="display:block;">
        <e-layers>
          <e-layer [shapeData]='shapeData' [polygonSettings]='polygonSettings'></e-
layer>
        </e-layers>
      </ejs-maps>
    </div>
  </div>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public polygonSettings?: object;
  ngOnInit(): void {
    this.polygonSettings = {
      tooltipSettings: { visible: true },
      polygons: [
        {
          tooltipTemplate: '<div style="width:200px;border: 2px solid
#000;padding: 10px;background-color:white;color:black;font-weight:bold;font-
size:15px;"> Country Name : Saudi Arabia</div>',
          points: [
            { longitude: 34.88539587371454, latitude: 28.181421087099537 },
            { longitude: 37.50029619722466, latitude: 24.299419888989462 },
            { longitude: 39.22241423764024, latitude: 22.638529461838658 },
            { longitude: 38.95650769309776, latitude: 21.424998160017495 },
            { longitude: 40.19963938650778, latitude: 20.271205391339606 },
            { longitude: 41.76547269134551, latitude: 18.315451049867193 },
            { longitude: 42.78452077838921, latitude: 16.097235052947966 },
          ]
        }
      ]
    }
  }
}
```

```

    { longitude: 43.36984949591576, latitude: 17.188572054533054 },
    { longitude: 44.12558191797012, latitude: 17.407258102232234 },
    { longitude: 46.69027032797584, latitude: 17.33342243475734 },
    { longitude: 47.09312386141585, latitude: 16.97087769526452 },
    { longitude: 48.3417299826302, latitude: 18.152700711188004 },
    { longitude: 49.74762591400318, latitude: 18.81544363931681 },
    { longitude: 52.41428026336621, latitude: 18.9035706497573 },
    { longitude: 55.272683129240335, latitude: 20.133861012918544 },
    { longitude: 55.60121336079203, latitude: 21.92042703112351 },
    { longitude: 55.08204399107967, latitude: 22.823302662258882 },
    { longitude: 52.743894337844154, latitude: 22.954463486477437 },
    { longitude: 51.47035908651375, latitude: 24.35818837668566 },
    { longitude: 51.122553219055874, latitude: 24.666679732426346 },
    { longitude: 51.58731671256831, latitude: 25.173806925822717 },
    { longitude: 51.35950585992913, latitude: 25.84556484481108 },
    { longitude: 51.088770529661275, latitude: 26.168494193631147 },
    { longitude: 50.78527056538036, latitude: 25.349051242147596 },
    { longitude: 50.88330288802666, latitude: 24.779242606720743 },
    { longitude: 50.19702490702369, latitude: 25.66825106363693 },
    { longitude: 50.066461167339924, latitude: 26.268905608606616 },
    { longitude: 49.645896067213215, latitude: 27.15116474192905 },
    { longitude: 48.917371072320805, latitude: 27.55738830340198 },
    { longitude: 48.3984720209192, latitude: 28.566207269716173 },
    { longitude: 47.68851714518985, latitude: 28.5938991332588 },
    { longitude: 47.45059089191449, latitude: 29.009321449856984 },
    { longitude: 44.73549453609391, latitude: 29.157358362696385 },
    { longitude: 41.79487372890989, latitude: 31.23489959729713 },
    { longitude: 40.36977176033773, latitude: 31.9642352513131 },
    { longitude: 39.168270913149826, latitude: 32.18348471414393 },
    { longitude: 37.019253492546454, latitude: 31.47710220862595 },
    { longitude: 37.99644645508337, latitude: 30.4851028633376 },
    { longitude: 37.67756530485232, latitude: 30.3636358598429 },
    { longitude: 37.50181466030105, latitude: 29.960155516804974 },
    { longitude: 36.700288181129594, latitude: 29.882136586478993 },
    { longitude: 36.100009274845206, latitude: 29.15308642012721 },
    { longitude: 34.85774476486728, latitude: 29.3103032832622 },
    { longitude: 34.64498583263142, latitude: 28.135787235699823 },
    { longitude: 34.88539587371454, latitude: 28.181421087099537 }
  ],
  fill: 'red',
  opacity: 0.7,
  borderColor: 'green',
  borderWidth: 2,
  borderOpacity: 0.7
}
]
};
this.shapeData = world_map;
}
}

```

MAIN.TS

```

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

```

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

Markers in Angular Maps component

Markers are notes that are used to leave a message on the Maps. It indicates or marks a specific location with desired symbols on the Maps. It can be enabled by setting the [visible](#) property of the [markerSettings](#) to **true**.

Adding marker

To add the markers, the [dataSource](#) property of the [markerSettings](#) has a list of objects that contains the data for markers. Using this property, any number of markers can be added to the layers of the Maps. By default, it displays the markers based on the specified latitude and longitude in the given data source. Each data source object contains the following list of properties.

- latitude - The latitude point which determines the X location of the marker.
- longitude - The longitude point which determines the Y location of the marker.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';

@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
    'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      height: 20,
      width: 20,
      dataSource: [
        { latitude: 49.95121990866204, longitude: 18.468749999999998
      },
        { latitude: 59.88893689676585, longitude: -109.3359375 },
        { latitude: -6.64607562172573, longitude: -55.54687499999999
      }
    ]
  }
}
```

```

        ],
        animationDuration: 0,
    }
}
}

```

MAIN.TS

```

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

```

Adding marker template

The Marker can be added as a template in the Maps component. The [template](#) property of the [markerSettings](#) is used to set the HTML string or id of an element as a template.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' >
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [
      {
        visible: true,
        template: '<div id="marker4" class="markerTemplate">Europe'
+
        '</div>',
        dataSource: [
          { latitude: 49.95121990866204, longitude:
18.468749999999998 }
        ],

```

```

        animationDuration: 0,
      },
      {
        visible: true,
        template: '<div id="marker5" class="markerTemplate"
style="width:50px">North America' +
          '</div>',
        dataSource: [
          { latitude: 59.88893689676585, longitude: -109.3359375 }
        ],
        animationDuration: 0
      },
      {
        visible: true,
        template: '<div id="marker6" class="markerTemplate"
style="width:50px">South America' +
          '</div>',
        dataSource: [
          { latitude: -6.64607562172573, longitude: -
55.54687499999999 }
        ],
        animationDuration: 0
      }
    ]
  }
}

```

MAIN.TS

```

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

```

Customization

The following properties are available in [markerSettings](#) to customize the Markers of the Maps component.

- [border](#) - To customize the color, width and opacity of the border for the markers in Maps.
- [fill](#) - To apply the color for markers in Maps.
- [dashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the markers in Maps.
- [height](#) - To customize the height of the markers in Maps.
- [width](#) - To customize the width of the markers in Maps.
- [offset](#) - To customize the position of the markers in Maps.
- [opacity](#) - To customize the transparency of the markers in Maps.
- [animationDelay](#) - To change the time delay in the transition for markers.
- [animationDuration](#) - To change the time duration of animation for markers.

APP.COMPONENT.TS

```

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

```

```

import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' >
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      border: {
        color: 'green',
        width: 2
      },
      fill: 'red',
      dashArray: '1',
      height: 20,
      width: 20,
      opacity: 0.9,
      animationDelay: 100,
      animationDuration: 1000,
      shape: 'Balloon',
      dataSource: [
        { latitude: 37.0000, longitude: -120.0000, city:
'California' },
        { latitude: 40.7127, longitude: -74.0059, city: 'New York'
},
        { latitude: 42, longitude: -93, city: 'Iowa' }
      ]
    }]
  }
}

```

MAIN.TS

```

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

```

Marker shapes

The Maps component supports the following marker shapes. To set the shape of the marker, the [shape](#) property in [markerSettings](#) is used.

- Balloon
- Circle
- Cross
- Diamond
- Image
- Rectangle
- Start
- Triangle
- VerticalLine
- HorizontalLine

Rendering marker shape as image

To render a marker as an image in Maps, set the [shape](#) property of [markerSettings](#) as **Image** and specify the path of the image to [imageUrl](#) property. There is another way to render a marker as an image using the [imageUrlValuePath](#) property of the [markerSettings](#). Bind the field name that contains the path of the image in the data source to the [imageUrlValuePath](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

```

        imageUrl:
'https://ej2.syncfusion.com/angular/demos/assets/maps/images/ballon.png',
        height: 20,
        width: 20,
        dataSource: [
            { latitude: 37.0000, longitude: -120.0000, city:
'California' },
            { latitude: 40.7127, longitude: -74.0059, city: 'New York'
},
            { latitude: 42, longitude: -93, city: 'Iowa' }
        ]
    }
}
}

```

MAIN.TS

```

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

```

Multiple marker groups

Multiple groups of markers can be added to the Maps using the [markerSettings](#) in which the properties of markers are added as an array. The customization for the markers can be done with the [markerSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' >
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
  }
}

```



```

    this.markerSettings = [{
      visible: true,
      shape: 'Diamond',
      height: 15,
      fill: "green",
      width: 15,
      dataSource: [
        {
          latitude: 37.0000, longitude: -120.0000,
name: 'California',
        },
        {
          latitude: 40.7127, longitude: -74.0059, name: "New York"
        },
        {
          latitude: 42, longitude: -93, name: 'Iowa'
        }
      ]
    },
    {
      visible: true,
      dataSource: [{
        latitude: 19.228825, longitude: 72.854118, name: "Mumbai"
      }, {
        latitude: 28.610001, longitude: 77.230003, name: "Delhi"
      }, {
        latitude: 13.067439, longitude: 80.237617, name: "Chennai"
      }],
      shape: 'Circle',
      fill: "blue",
      height: 10,
      width: 10
    }
  ]
}

```

MAIN.TS

```

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

```

Customize marker shapes from data source

Bind different colors and shapes to the marker from data source

Using the [shapeValuePath](#) and [colorValuePath](#) properties, the color and shape of the marker can be applied from the given data source. Bind the data source to the [dataSource](#) property of the [markerSettings](#) and set the field names that contains the shape and color values in the data source to the [shapeValuePath](#) and [colorValuePath](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'

```

```

import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      shapeValuePath: 'shape',
      colorValuePath: 'color',
      dataSource: [
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name: 'Europe', color: 'red', shape: 'Triangle' },
        { latitude: 59.88893689676585, longitude: -109.3359375,
name: 'North America',
        color: 'blue', shape: 'Pentagon' },
        { latitude: -6.64607562172573, longitude: -
55.546874999999999, name: 'South America',
        color: 'green', shape: 'InvertedTriangle' }
      ]
    }];
  }
}

```

MAIN.TS

```

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

```

Setting value path from the data source

The latitude and longitude values are used to determine the location of each marker in the Maps. The [latitudeValuePath](#) and [longitudeValuePath](#) properties are used to specify the value path that presents in the data source of the marker. In the following example, the field name from the data source is set to the [latitudeValuePath](#) and [longitudeValuePath](#) properties.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      latitudeValuePath: 'latitude',
      longitudeValuePath: 'longitude',
      dataSource: [
        { latitude: 49.95121990866204, longitude: 18.468749999999998
      },
        { latitude: 59.88893689676585, longitude: -109.3359375},
        { latitude: -6.64607562172573, longitude: -55.54687499999999
      }
    ]
  }
}

```

MAIN.TS

```

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

```

Repositioning the marker using drag and drop

The markers on the map can be dragged and dropped to change their position. To enable marker drag and drop, set the [enableDrag](#) property to **true** in the [markerSettings](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, MapsTooltipService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container'>
  <e-layers>
    <e-layer [shapeSettings] = 'shapeSettings' [shapeData]= 'shapeData'
    [markerSettings] = 'markerSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  public shapeSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeSettings = {
      fill: '#C3E6ED'
    };
    this.markerSettings = [{
      enableDrag: true,
      dataSource: [
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name: 'MarkerOne' },
        { latitude: 59.88893689676585, longitude: -109.3359375,
name: 'MarkerTwo' },
        { latitude: -6.64607562172573, longitude: -
55.546874999999999 , name: 'MarkerThree' },
        { latitude: 23.644385824912135, longitude:
77.83189239539234 , name: 'MarkerFour' },
        { latitude: 63.66569332894224, longitude: 98.2225173953924
, name: 'MarkerFive' }
      ],
      visible: true,
      animationDuration: 0,
      shape: 'Balloon',
      width: 20,
      height: 20,
      border: { width: 2, color: '#285255' },
      tooltipSettings: {
        visible: true,
        valuePath: 'name',
      }
    }
  ]
}
}

```

MAIN.TS

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

The data of the drag and dropped marker in the marker data source can be customized using the [markerDragStart](#) and [markerDragEnd](#) events. When you change the appropriate marker data, the tooltip and legend item text of that marker are automatically updated. The following properties are available in the event argument of the marker drag events.

Argument Name	Description
dataIndex	It represents the index of the data of the dragged marker in the marker data source.
latitude	It represents the latitude coordinate point of the dragged marker.
longitude	It represents the longitude coordinate point for the dragged marker.
markerIndex	It represents the index of the marker setting.
layerIndex	It represents the index of the layer in which the marker belongs.
name	It represents the name of the event.
x	It represents the horizontal location of the mouse pointer on the map when the drag action is performed.
y	It represents the vertical location of the mouse pointer on the map when the drag action is performed.

The following example shows how to use marker drag events to customize the data of the drag and dropped marker in the marker data source.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, MarkerService, MapsTooltipService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit, ViewChild } from '@angular/core';
import { Maps, IMarkerDragEventArgs } from '@syncfusion/ej2-angular-maps';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, MarkerService, MapsTooltipService],
```

```

standalone: true,
  selector: 'app-container',
  template: `
    <ejs-maps id="rn-container" #maps
(markerDragStart)="markerDragStart($event)"
(markerDragEnd)="markerDragEnd($event)" [legendSettings]="legendSettings">
      <e-layers>
        <e-layer [shapeSettings]="shapeSettings" [shapeData]="shapeData"
[markerSettings]="markerSettings"></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  @ViewChild('maps')
  public mapsInstance?: Maps;
  public shapeData?: object;
  public markerSettings?: object;
  public shapeSettings?: object;
  public zoomSettings?: object;
  public markerDragStart = (args: IMarkerDragEventArgs | any) => {
    // When the marker begins to move on the map, the event is triggered.
  };
  public markerDragEnd = (args: IMarkerDragEventArgs | any) => {
    // When the marker on the map stops dragging, the event is triggered.
    (this.mapsInstance as any).layers[args.layerIndex].markerSettings[args.markerIndex].dataSource[args.dataIndex].name = 'Dragged Marker ' + (args.dataIndex + 1);
    (this.mapsInstance as Maps).refresh();
  };
  public legendSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.legendSettings = {
      visible: true,
      type: 'Markers',
      shape: 'Circle',
      shapeWidth: 10,
      shapeHeight: 10,
      fill: '#FF471A',
      shapeBorder: { width: 2, color: '#285255' },
    };
    this.shapeSettings = {
      fill: '#C3E6ED',
    };
    this.zoomSettings = {
      enable: false,
    };
    this.markerSettings = [
      {
        enableDrag: true,
        legendText: 'name',
        dataSource: [
          {
            latitude: 49.95121990866204,
            longitude: 18.468749999999998,
            name: 'MarkerOne',
          },
        ],
      },
    ],
  },

```

```

        {
          latitude: 59.88893689676585,
          longitude: -109.3359375,
          name: 'MarkerTwo',
        },
        {
          latitude: -6.64607562172573,
          longitude: -55.54687499999999,
          name: 'MarkerThree',
        },
        {
          latitude: 23.644385824912135,
          longitude: 77.83189239539234,
          name: 'MarkerFour',
        },
        {
          latitude: 63.66569332894224,
          longitude: 98.2225173953924,
          name: 'MarkerFive',
        },
      ],
      visible: true,
      animationDuration: 0,
      shape: 'Balloon',
      width: 20,
      height: 20,
      border: { width: 2, color: '#285255' },
      tooltipSettings: {
        visible: true,
        valuePath: 'name',
      },
    },
  ];
}

```

MAIN.TS

```

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

```

Marker zooming

The Maps can be initially scaled to the center value based on the marker distance. This can be achieved by setting the [shouldZoomInitially](#) property in [zoomSettings](#) as **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';

```

```

@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [zoomSettings] ='zoomSettings'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  public zoomSettings?: object
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [
      {
        visible: true,
        dataSource: [
          { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
          { latitude: 59.88893689676585, longitude: -109.3359375,
name:'North America' },
          { latitude: -6.64607562172573, longitude: -
55.54687499999999, name:'South America'}
        ],
      },
    ];
    this.zoomSettings = {
      enable: true,
      horizontalAlignment:'Near',
      shouldZoomInitially: true
    }
  }
}

```

MAIN.TS

```

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

```

Marker clustering

Maps provide support to cluster the markers when they overlap each other. The number on a cluster indicates how many overlapped markers it contains. If zooming is performed on any of the cluster locations in Maps, the number on the cluster will decrease, and the individual markers will be seen on

the map. When zooming out, the overlapping marker will increase. So that it can cluster again and increase the count over the cluster.

To enable clustering in markers, set the [allowClustering](#) property of [markerClusterSettings](#) as **true** and customization of clustering can be done with the [markerClusterSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, MapsTooltipService, ZoomService } from
 '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, ViewChild, OnInit } from
 '@angular/core';
import { world_map } from './world-map';
import { cluster } from './marker-location';;
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

```

    }
  }];
}
}

```

MAIN.TS

```

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

```

Customization of marker cluster

The following properties are available to customize the marker clustering in the Maps component.

- [border](#) - To customize the color, width and opacity of the border of cluster in Maps.
- [connectorLineSettings](#) - To customize the connector line in cluster separating the markers.
- [dashArray](#) - To customize the dash array for the marker cluster in Maps.
- [fill](#) - Applies the color of the cluster in Maps.
- [height](#) - To customize the height of the marker cluster in Maps.
- [imageUrl](#) - To customize the URL path for the marker cluster when the cluster shape is set as image in Maps.
- [labelStyle](#) - To customize the text in marker cluster.
- [offset](#) - To customize the offset position for the marker cluster in Maps.
- [opacity](#) - To customize the opacity of the marker cluster.
- [shape](#) - To customize the shape for the cluster of markers.
- [width](#) - To customize the width of the marker cluster in Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, MapsTooltipService, ZoomService } from
 '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { Point } from '@syncfusion/ej2-angular-maps';
import { world_map } from './world-map';
import { cluster } from './marker-location';;
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings' [layers] =
    'layers'>
    </ejs-maps>`
})
export class AppComponent implements OnInit {

```

```

public zoomSettings?: object;
public layers?: object;
ngOnInit(): void {
    this.zoomSettings = {
        enable: true
    };
    this.layers = [{
        shapeData: world_map,
        shapeSettings: { fill: '#C1DFF5' },
        markerClusterSettings: {
            allowClustering: true,
            allowClusterExpand: true,
            shape: 'Circle',
            height: 40,
            width: 40,
            labelStyle : { color: 'white' },
            offset: new Point(10, 20),
            opacity: 0.9,
            fill: 'green',
            connectorLineSettings: {
                color: 'orange',
                opacity: 0.8,
                width: 2
            }
        },
        markerSettings: [{
            dataSource: cluster,
            visible: true,
            animationDuration: 0,
            shape: 'Balloon',
            height: 20,
            width: 20,
            tooltipSettings: {
                visible: true,
                valuePath: 'area',
            }
        }]
    }];
}
}

```

MAIN.TS

```

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

```

Expanding the marker cluster

The cluster is formed by grouping an identical and non-identical marker from the surrounding area. By clicking on the cluster and setting the [allowClusterExpand](#) property in [markerClusterSettings](#) as **true** to expand the identical markers. If zooming is performed in any of the locations of the cluster, the number on the cluster will decrease and the overlapping marker will be split into an individual marker on the map. When performing zoom out, it will increase the marker count and then cluster it again.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [zoomSettings] ='zoomSettings' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] = 'markerSettings'
    [markerClusterSettings] = 'markerClusterSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  public zoomSettings?: object;
  public markerClusterSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      dataSource: [
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name:'Europe' },
        { latitude: 59.88893689676585, longitude: -109.3359375,
name:'North America' },
        { latitude: -6.64607562172573, longitude: -
55.546874999999999, name:'South America'}
      ]
    }];
    this.zoomSettings = {
      enable: true,
      mouseWheelZoom : true,
    },
  },

```

```

        this.markerClusterSettings = {
            allowClustering: true,
            allowClusterExpand: true,
            shape: 'Circle',
            height: 40,
            width: 40,
            labelStyle : { 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));

```

Tooltip for marker

Tooltip is used to display more information about a marker on mouse over or touch end event. This can be enabled separately for marker by setting the [visible](#) property of [tooltipSettings](#) to [Link to the Video](#). The [valuePath](#) property in the [tooltipSettings](#) takes the field name that presents in data source and displays that value as tooltip text.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { MarkerService, MapsTooltipService } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
    'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = usa_map;
    this.markerSettings = [{
      dataSource: [

```

```

        { latitude: 40.7424509, longitude: -74.0081468, city: 'New
York' }
    ],
    visible:true,
    shape:'Circle',
    fill:'white',
    width:3,
    animationDuration:0,
    border: { width:2, color:'green'},
    tooltipSettings: {
        visible: true,
        valuePath:'city'
    }
  }]
}
}

```

MAIN.TS

```

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

```

This section shows how to add and customize bubbles in the Maps component using the video below.

Bubble in Angular Maps component

Bubbles in the Maps component represent the underlying data values of the Maps. It can be scattered throughout the Maps shapes that contain values in the data source. Bubbles are enabled by setting the [visible](#) property of [bubbleSettings](#) to **true**. To add bubbles to the Maps, bind the data source to the [dataSource](#) property of [bubbleSettings](#) and set the field name, that contains the numerical data, in the data source to the [valuePath](#) property.

`typescript

export let world_map = // paste the World map from WorldMap.json GeoJSON file.

,

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>

```

```

    <e-layers>
      <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
        'bubbleSettings'></e-layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 20,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'New Zealand', population: '19651127' },
        { name: 'Pakistan', population: '3090416' }
      ],
      maxRadius: 40,
      valuePath: 'population'
    }]
  }
}

```

MAIN.TS

```

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

```

Bubble shapes

The following types of shapes are available to render the bubbles in Maps.

- Circle
- Square

By default, bubbles are rendered in the **Circle** type. To change the type of the bubble, set the [bubbleType](#) property of [bubbleSettings](#) property as **Square** to render the square shape bubbles.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({

```

```

imports: [
    MapsModule
],
providers: [BubbleService],
standalone: true,
selector: 'app-container',
template:
`<ejs-maps id='rn-container'>
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
'bubbleSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      bubbleType: 'Square',
      visible: true,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'Pakistan', population: '3090416' }
      ],
      valuePath: 'population'
    }]
  }
}

```

MAIN.TS

```

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

```

Customization

The following properties are available in [bubbleSettings](#) to customize the bubbles of the Maps component.

- [border](#) – To customize the color, width and opacity of the border of the bubbles in Maps.
- [fill](#) – To apply the color for bubbles in Maps.
- [opacity](#) – To apply opacity to the bubbles in Maps.
- [animationDelay](#) - To change the time delay in the transition for bubbles.
- [animationDuration](#) - To change the time duration of animation for bubbles.

APP.COMPONENT.TS


```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService],
  standalone: true,
  selector: 'app-container',
  template: `
    <ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
'bubbleSettings'></e-layer>
      </e-layers>
    </ejs-maps>
  `
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 5,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'New Zealand', population: '19651127' },
        { name: 'Pakistan', population: '3090416' }
      ],
      fill: 'green',
      animationDelay: 100,
      animationDuration: 1000,
      maxRadius: 40,
      border: {
        color: 'blue',
        width: 2
      },
      opacity: 1,
      valuePath: 'population'
    }]
  }
}

```

MAIN.TS

```

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

```

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

Setting colors to the bubbles from the data source

The color for each bubble in the Maps can be set using the [colorValuePath](#) property of [bubbleSettings](#). The value for the [colorValuePath](#) property is the field name from the data source of the [bubbleSettings](#) which contains the color values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
    'bubbleSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 20,
      colorValuePath: 'color',
      dataSource: [
        { name: 'India', population: '38332521', color: 'blue' },
        { name: 'New Zealand', population: '19651127', color:
        '#c2d2d6' },
        { name: 'Pakistan', population: '3090416', color: '#09156d'
      }
    ]
  },
  {
    maxRadius: 40,
    valuePath: 'population'
  }
  ]
}
```

```
}

```

MAIN.TS

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

Setting the range of the bubble size

The size of the bubbles is calculated from the values got from the [valuePath](#) property. The range for the radius of the bubbles can be modified using [minRadius](#) and [maxRadius](#) properties.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
    'bubbleSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name',
    this.bubbleSettings = [{
      visible: true,
      minRadius: 5,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'New Zealand', population: '19651127' },
        { name: 'Pakistan', population: '3090416' }
      ],
      maxRadius: 80,
      valuePath: 'population'
    }]
```

```

    ]]
  }
}

```

MAIN.TS

```

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

```

Multiple bubble groups

Multiple groups of bubbles can be added to the Maps using the [bubbleSettings](#) in which the properties of bubbles are added as an array. The customization for the bubbles can be done with the [bubbleSettings](#). In the following example, the gender-wise population ratio is demonstrated with two different bubble groups.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
    'bubbleSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 5,
      valuePath: "femaleRatio",
      colorValuePath: "femaleRatioColor",

```

```

        dataSource: [
            {
                country: "United States", femaleRatio: 50.50442726,
maleRatio: 49.49557274, femaleRatioColor: "green", maleRatioColor: "blue"
            },
            {
                country: "India", femaleRatio: 48.18032713, maleRatio:
51.81967287, femaleRatioColor: "blue", maleRatioColor: "#c2d2d6"
            },
            {
                country: "Oman", femaleRatio: 34.15597234, maleRatio:
65.84402766, femaleRatioColor: "#09156d", maleRatioColor: "orange"
            },
            {
                country: "United Arab Emirates", femaleRatio:
27.59638942, maleRatio: 72.40361058, femaleRatioColor: "#09156d",
maleRatioColor: "orange"
            }
        ],
        maxRadius: 20,
    },
    {
        visible: true,
        bubbleType: 'Circle',
        opacity: 0.4,
        minRadius: 15,
        valuePath: "maleRatio",
        colorValuePath: "maleRatioColor",
        dataSource: [
            {
                country: "United States", femaleRatio: 50.50442726,
maleRatio: 49.49557274, femaleRatioColor: "green", maleRatioColor: "blue"
            },
            {
                country: "India", femaleRatio: 48.18032713, maleRatio:
51.81967287, femaleRatioColor: "blue", maleRatioColor: "#c2d2d6"
            },
            {
                country: "Oman", femaleRatio: 34.15597234, maleRatio:
65.84402766, femaleRatioColor: "#09156d", maleRatioColor: "orange"
            },
            {
                country: "United Arab Emirates", femaleRatio:
27.59638942, maleRatio: 72.40361058, femaleRatioColor: "#09156d",
maleRatioColor: "orange"
            }
        ],
        maxRadius: 25,
    }
]
}
}

```

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 for bubble

The tooltip for the bubbles can be enabled by setting the [visible](#) of the [tooltipSettings](#) as **true**. The content for the tooltip can be set using the [valuePath](#) property in the [tooltipSettings](#) of the [bubbleSettings](#) where the value for the [valuePath](#) property is the field name from the data source of the [bubbleSettings](#). Also added any HTML element as the template in tooltip using the [template](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService, BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService, BubbleService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapeDataPath]='shapeDataPath'
        [shapePropertyPath]='shapePropertyPath' [bubbleSettings] =
        'bubbleSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.bubbleSettings = [{
      visible: true,
      minRadius: 5,
      dataSource: [
        { name: 'India', population: '38332521' },
        { name: 'New Zealand', population: '19651127' },
        { name: 'Pakistan', population: '3090416' }
      ],
      maxRadius: 80,
      valuePath: 'population',
      tooltipSettings: {
        visible: true,
        valuePath: 'population'
      }
    }]
  }
}
```

```

    }
  ]]
}
}

```

MAIN.TS

```

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

```

Legend in Angular Maps component

A Legend is a visual representation of the symbols used on the Maps. It can be represented in various colors, shapes or other identifiers based on the data and provides valuable information for interpreting what the Maps are displaying. It explains what each symbol in the Maps represents. Legends are enabled by setting the [visible](#) property of [legendSettings](#) property to **true**.

Modes of legend

Legend had two types of mode.

1. **Default** mode
2. **Interactive** mode

Default mode

Default mode legends having symbols with legend labels, used to identify the shape or bubble or marker color. To enable this option by setting the [mode](#) property of [legendSettings](#) as **Default**.

Interactive mode

The legends can be made interactive with an arrow mark indicating the exact range color in the legend when the mouse hovers over the corresponding shapes. To enable this type of mode by setting the [mode](#) property of [legendSettings](#) as **Interactive**. The [invertedPointer](#) property is used to enable or disable the visibility of the inverted pointer in interactive legend in Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>

```

```

    <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
    'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
    'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
      { "Country": "Poland", "Membership": "Non-Permanent" },
      { "Country": "Sweden", "Membership": "Non-Permanent" }
    ];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
      colorValuePath: 'Membership',
      colorMapping: [
        {
          value: 'Permanent', color: '#D84444'
        },
        {
          value: 'Non-Permanent', color: '#316DB5'
        }
      ]
    };
    this.legendSettings = {
      visible: true,
      mode: 'Interactive',
      invertedPointer: true
    }
  }
}

```

MAIN.TS

```

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

```

Positioning of the legend

The legend can be positioned in the following two ways:

- Absolute position
- Dock position

Absolute position

The legend of the Maps can be positioned using the [location](#) property in the [legendSettings](#) property. For positioning the legend based on co-ordinates corresponding to a Maps, the [position](#) property is set as **Float**.

Dock position

Legends are positioned in the following locations within the container. The [position](#) property in [legendSettings](#) property is used to set these options in Maps.

- Top
- Left
- Bottom
- Right

The above four positions can be aligned with combination of **Near**, **Center** and **Far** using [alignment](#) property in [legendSettings](#) property. So, the legend can be aligned to 12 positions.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
        'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
        'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
    ]
  }
}
```

```

        { "Country": "Poland", "Membership": "Non-Permanent" },
        { "Country": "Sweden", "Membership": "Non-Permanent" } ]];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
            {
                value: 'Permanent', color: '#D84444'
            },
            {
                value: 'Non-Permanent', color: '#316DB5'
            }
        ]
    };
    this.legendSettings = {
        visible: true,
        position: 'Top',
        alignment: 'Near'
    }
}
}

```

MAIN.TS

```

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

```

Legend for shapes

Legend for shapes can be generated from color mapping types such as equal color mapping, range color mapping and desaturation color mapping.

The below code snippet demonstrate the equal color mapping legends for the shapes. To bind the given data source to the [dataSource](#) property of [layerSettings](#) property. Set the value of [shapePropertyPath](#) to **name** and [shapeDataPath](#) to **Country**. To enable equal color mapping, set the [colorMapping](#) as an array in [shapeSettings](#) property. Finally, set the [visible](#) property of [legendSettings](#) as **true**. The [label](#) property in [colorMapping](#) property is used to set the text name for legend in Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',

```

```

    template:
      `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
        <e-layers>
          <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
            'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
            'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
        </e-layers>
      </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public dataSource?: object[];
    public shapeData?: object;
    public shapePropertyPath?: string;
    public shapeDataPath?: string;
    public shapeSettings?: object;
    public legendSettings?: object;
    ngOnInit(): void {
      this.dataSource = [
        { "Country": "China", "Membership": "Permanent" },
        { "Country": "France", "Membership": "Permanent" },
        { "Country": "Russia", "Membership": "Permanent" },
        { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
        { "Country": "Poland", "Membership": "Non-Permanent" },
        { "Country": "Sweden", "Membership": "Non-Permanent" }
      ];
      this.shapeData = world_map;
      this.shapePropertyPath = 'name';
      this.shapeDataPath = 'Country';
      this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
          {
            value: 'Permanent', color: '#D84444'
          },
          {
            value: 'Non-Permanent', color: '#316DB5'
          }
        ]
      };
      this.legendSettings = {
        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));

```

Legend shape

Maps supports the following types of legend shapes. The [shape](#) property in the [legendSettings](#) property can be used to change the type of legend shapes.

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

The shape of legends can be customized using the [shapeWidth](#), [shapeHeight](#), [shapeBorder](#) and [shapePadding](#) properties.

Customization

The following properties are available in legend to customize the legend shape and legend text in Maps.

- [background](#) - To customize the background color of the Legend.
- [border](#) - To customize the color, width and opacity of the border for the Legend.
- [fill](#) - To apply the color for the Legend.
- [labelDisplayMode](#) - To customize the display mode for the Legend text.
- [labelPosition](#) - To customize the position of the Legend text.
- [orientation](#) - To customize the orientation of the Legend.
- [textStyle](#) - To customize the text style for Legend.
- [title](#) - To apply the title for the Legend.
- [titleStyle](#) - To customize the style of the title for the Legend.
- [height](#) - To customize the height of the Legend.
- [width](#) - To customize the width of the Legend.
- [opacity](#) - To apply the opacity to the Legend.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
      'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
      'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>`
})
```

```

</ejs-maps>`
}))
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
      { "Country": "Poland", "Membership": "Non-Permanent" },
      { "Country": "Sweden", "Membership": "Non-Permanent" }
    ];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
      colorValuePath: 'Membership',
      colorMapping: [
        {
          value: 'Permanent', color: '#D84444'
        },
        {
          value: 'Non-Permanent', color: '#316DB5'
        }
      ]
    };
    this.legendSettings = {
      visible: true,
      background: 'green',
      border: {
        color: 'blue',
        width: 2
      },
      fill: 'orange',
      labelPosition: 'Before',
      orientation: 'Vertical',
      textStyle: {
        size: '12px',
        color: 'red',
        fontStyle: 'italic'
      },
      title: {
        description: 'Legend title',
        text: 'Legend'
      },
      titleStyle: {
        size: '12px',
        color: '#d6e341',
        fontStyle: 'italic'
      }
    }
  }
}

```

MAIN.TS

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

Legend for items excluded from color mapping

The legend can be enabled for items excluded from the color mapping using the [color](#) property in [colorMapping](#). Refer to the **Population_density** data which demonstrates the population density of some countries.

```
`typescript
```

```
export let Population_density = [
```

```
  https://ej2.syncfusion.com/angular/documentation.
```

```
{
```

```
  'code': 'AE',
```

```
  'value': 90,
```

```
  'name': 'United Arab Emirates',
```

```
  'population': 8264070,
```

```
  'density': 99
```

```
},
```

```
{
```

```
  'code': 'GB',
```

```
  'value': 257,
```

```
  'name': 'United Kingdom',
```

```
  'population': 62041708,
```

```
  'density': 255
```

```
},
```

```
{
```

```
  'code': 'US',
```

```
  'value': 34,
```

```
  'name': 'United States',
```

```
  'population': 325020000,
```

```
  'density': 33
```

```
}
```

```
...
```

```
];
`
```

In the following example, color mapping is added for the ranges from **0** to **200**. If there are any records in the data source that are outside of this range, the color mapping will not be applied. To apply the color for these excluded items, set the [color](#) property alone in the [colorMapping](#). To enable legend for these items, set the [visible](#) property of [legendSettings](#) to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { Population_Density } from './data'
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

```

        },
    ],
};
this.legendSettings = {
    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));

```

Hide desired legend items

Use the [showLegend](#) in the [colorMapping](#) property to show or hide the desired legend items in Maps. If the [showLegend](#) property is set to **false**, the legend item will be hidden. otherwise, it will be visible.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```



```

        { "Country": "Russia", "Membership": "Permanent" },
        { "Country": "Kazakhstan", "Membership": "Both" },
        { "Country": "Poland", "Membership": "Non-Permanent" },
        { "Country": "Sweden", "Membership": "Non-Permanent" } ] ];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
            {
                value: 'Permanent', color: '#D84444', showLegend : true
            },
            {
                value: 'Non-Permanent', color: '#316DB5', showLegend :
false
            },
            {
                color: 'violet', showLegend : false
            }
        ]
    };
    this.legendSettings = {
        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));

```

Hide legend items based data source value

Depending on the boolean values provided in the data source, the legend items will be hidden or visible. Bind the field name that contains the visibility state in the data source to the [showLegendPath](#) property of the [legendSettings](#) to achieve this.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
Maps.Inject(Legend);
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',

```

```

    template:
      `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
        <e-layers>
          <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
            'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
            'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
        </e-layers>
      </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public dataSource?: object[];
    public shapeData?: object;
    public shapePropertyPath?: string;
    public shapeDataPath?: string;
    public shapeSettings?: object;
    public legendSettings?: object;
    ngOnInit(): void {
      this.dataSource = [
        { "Country": "China", "Membership": "Permanent", "visibility" :
"true", "color" : "red"},
        { "Country": "France", "Membership": "Permanent", "visibility" :
"true", "color" : "blue" },
        { "Country": "Russia", "Membership": "Permanent", "visibility" :
"true", "color" : "green"},
        { "Country": "Kazakhstan", "Membership": "Both", "visibility" :
"true", "color" : "orange"},
        { "Country": "Poland", "Membership": "Non-Permanent",
"visibility" : "true", "color" : "yellow"},
        { "Country": "Sweden", "Membership": "Non-Permanent",
"visibility" : "true", "color" : "pink"}];
      this.shapeData = world_map;
      this.shapePropertyPath = 'name';
      this.shapeDataPath = 'Country';
      this.shapeSettings = {
        colorValuePath: 'color',
      };
      this.legendSettings = {
        visible: true,
        showLegendPath: 'visibility'
      }
    }
  }
}

```

MAIN.TS

```

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

```

Binding legend item text from data source

To show the legend text based on values provided in the data source, use the [valuePath](#) property in the [legendSettings](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent", "visibility" :
"true", "color" : "red"},
      { "Country": "France", "Membership": "Permanent", "visibility" :
"true", "color" : "blue" },
      { "Country": "Russia", "Membership": "Permanent", "visibility" :
"true", "color" : "green"},
      { "Country": "Kazakhstan", "Membership": "Both", "visibility" :
"true", "color" : "orange"},
      { "Country": "Poland", "Membership": "Non-Permanent",
"visibility" : "true", "color" : "yellow"},
      { "Country": "Sweden", "Membership": "Non-Permanent",
"visibility" : "true", "color" : "pink"}];
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
      colorValuePath: 'color',
    };
    this.legendSettings = {
      visible: true,
      valuePath: 'Country'
    }
  }
}

```

MAIN.TS

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

Hide duplicate legend items

To hide the duplicate legend items in Maps, set the [removeDuplicateLegend](#) property to **true** in the [legendSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
      'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
      'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent", "visibility" :
      "true", "color" : "red"},
      { "Country": "France", "Membership": "Permanent", "visibility" :
      "true", "color" : "blue" },
      { "Country": "Russia", "Membership": "Permanent", "visibility" :
      "true", "color" : "green"},
      { "Country": "Kazakhstan", "Membership": "Both", "visibility" :
      "true", "color" : "orange"},
      { "Country": "Poland", "Membership": "Non-Permanent",
      "visibility" : "true", "color" : "yellow"},
      { "Country": "Sweden", "Membership": "Non-Permanent",
      "visibility" : "true", "color" : "pink"}];
```

```

    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'Country';
    this.shapeSettings = {
      colorValuePath: 'color',
    };
    this.legendSettings = {
      visible: true,
      valuePath: 'Membership',
      removeDuplicateLegend: true
    }
  }
}

```

MAIN.TS

```

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

```

Toggle option in legend

The toggle option has been provided for legend. If the legend can be toggled, the given color will be changed to the corresponding Maps shape item. To enable the toggle options in Legend, set the [enable](#) property of the [toggleLegendSettings](#) to **true**.

The following properties are available to customize the toggle option in legend.

- [applyShapeSettings](#) – To apply a [fill](#) property color to the shapes when toggling the legend items.
- [fill](#) - To apply the color to the shape of the Maps for which legend item is toggled.
- [opacity](#) – To customize the transparency for the shapes for which legend item is toggled.
- [border](#) – To customize the color, width and opacity of the border of the shapes in Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { Population_Density } from './data'
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings] = 'legendSettings'>
    <e-layers>

```

```

    <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
    'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
    'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public dataSource?: object[];
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.dataSource = Population_Density;
    this.shapeData = world_map;
    this.shapePropertyPath = 'name';
    this.shapeDataPath = 'name';
    this.shapeSettings = {
      colorValuePath: 'density',
      colorMapping: [
        {
          from: 0, to: 100, color: 'rgb(153,174,214)',
        },
        {
          from: 101, to: 200, color: 'rgb(115,143,199)',
        },
        {
          color: 'rgb(77,112,184)'
        }
      ]
    };
    this.legendSettings = {
      visible: true,
      toggleLegendSettings: {
        enable: true,
        applyShapeSettings: false,
        border: {
          color: 'green',
          width: 2
        }
      }
    }
  }
}
}

```

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 for bubbles

To enable the legend for the bubble by setting the [visible](#) property of [legendSettings](#) as **true** and [type](#) property of [legendSettings](#) as **Bubbles**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, BubbleService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, BubbleService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-maps id='rn-container' [legendSettings]
='legendSettings' >
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
'bubbleSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: object | any;
  public shapePropertyPath?: object | any;
  public bubbleSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name',
    this.shapePropertyPath = 'name',
    this.bubbleSettings = [{
      visible: true,
      minRadius: 20,
      dataSource: [
        {color: 'green', name: 'India', population: '38332521' },
        {color: 'purple', name: 'New Zealand', population:
'19651127' },
        {color: 'blue', name: 'Pakistan', population: '3090416' }
      ],
      maxRadius: 40,
      valuePath: 'population',
      colorValuePath: 'color'
    }];
    this.legendSettings = {
      visible: true,
      type: 'Bubbles'
    };
  }
}

```

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 for markers

To enable legend for marker by setting the [visible](#) property of [legendSettings](#) as **true** and [type](#) property of [legendSettings](#) as **Markers**. The [legendText](#) property in the [markerSettings](#) can be used to show the legend text based on values provided in the data source.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```


MAIN.TS

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

Imitate/Map marker shape to the legend shape

To imitate or map the marker shape with its legend item shape, set the [useMarkerShape](#) property to **true** in the [legendSettings](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { markerDataSource } from './markerdata';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, MarkerService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps
      id="rn-container"
      style="display:block"
      [legendSettings]="legendSettings"
    >
      <e-layers>
        <e-layer [shapeData]= 'shapeData'
          [shapePropertyPath]="shapePropertyPath"
          [shapeDataPath]="shapeDataPath"
          [markerSettings]="markerSettings"
          [shapeSettings]="shapeSettings"
        ></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public shapeDataPath?: string;
  public shapePropertyPath?: string;
  public shapeSettings?: object;
  public markerSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.shapeDataPath = 'name';
    this.shapePropertyPath = 'name';
    this.legendSettings = {
      visible: true,
```

```

    type: 'Markers',
    useMarkerShape: true,
    toggleLegendSettings: {
      enable: true,
      applyShapeSettings: false,
      border: {
        color: 'green',
        width: 2,
      },
    },
  },
];
this.shapeSettings = {
  fill: '#E5E5E5',
};
this.markerSettings = [
  {
    dataSource: markerDataSource,
    colorValuePath: 'color',
    shapeValuePath: 'shape',
    legendText: 'country',
    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));

```

Navigation line in Angular Maps component

The navigation lines are used to denote the path between two locations. This feature can be used to draw flight or sea routes. Navigation lines are enabled by setting the [visible](#) property of the [navigationLineSettings](#) to **true**.

Customization

The following properties are available in [navigationLineSettings](#) to customize the navigation line of the Maps component.

- [color](#) - To apply the color for navigation lines in Maps.
- [dashArray](#) - To define the pattern of dashes and gaps that is applied to the outline of the navigation lines.
- [width](#) - To customize the width of the navigation lines.
- [angle](#) - To customize the angle of the navigation lines.
- [highlightSettings](#) - To customize the highlight settings of the navigation line.
- [selectionSettings](#) - To customize the selection settings of the navigation line.

To navigate the line between two cities on the world map, [latitude](#) and [longitude](#) values are used to indicate the start and end points of navigation lines drawn on Maps.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { NavigationLineService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [NavigationLineService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [navigationLineSettings] =
    'navigationLineSettings' ></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public navigationLineSettings?: object[];
  ngOnInit(): void {
    this.shapeData = world_map;
    this.navigationLineSettings = [{
      visible: true,
      latitude: [37.6276571, -122.4276688],
      longitude: [-74.0060, -117.7418381],
      color: 'black',
      angle: 90,
      width: 2,
      dashArray: '4'
    }]
  };
}

```

MAIN.TS

```

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

```

Enabling the arrows

To enable the arrow in the navigation line, set the [showArrow](#) property of [arrowSettings](#) to **true**. The following properties are available in [arrowSettings](#) to customize the arrow of the navigation lines.

- [color](#) - To apply the color for arrow of the navigation line.
- [offset](#) - To customize the offset position of the navigation line's arrow.

- [position](#) - To customize the position of the arrow in navigation line. The possible values can be **Start** and **End**.
- [size](#) - To customize the size of the arrow in pixels.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { NavigationLineService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [NavigationLineService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' >
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [navigationLineSettings] =
    'navigationLineSettings' ></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public navigationLineSettings?: object[];
  ngOnInit(): void {
    this.shapeData = world_map;
    this.navigationLineSettings = [{
      visible: true,
      latitude: [37.6276571, -122.4276688],
      longitude: [-74.0060, -117.7418381],
      color: 'black',
      angle: 90,
      width: 2,
      dashArray: '4',
      arrowSettings: {
        showArrow: true,
        size: 15,
        position: 'Start'
      }
    }]
  };
}
```

MAIN.TS

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

User interactions in Angular Maps component

Zooming

The zooming feature is used to zoom in and out of Maps to show in-depth information. It is controlled by the [zoomFactor](#) property of the [zoomSettings](#). The [zoomFactor](#) is increased or decrease dynamically based on zoom in and out interaction.

Enable zooming

Zooming of Maps is enabled by setting the [enable](#) property of [zoomSettings](#) to **true**.

<!-- markdownlint-disable MD010 -->

Enable panning

To enable the panning feature, set the [enablePanning](#) property of [zoomSettings](#) to **true**.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

MAIN.TS

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

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

app.module.ts

Injecting ZoomService into NgModule.

`typescript

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

Various type of zooming

Zooming can be performed in following types:

Zooming toolbar

The following options are available in toolbar.

1. Zoom - Provides rectangular zoom support.
2. ZoomIn - Zooms in the Maps.
3. ZoomOut - Zooms out the Maps.
4. Pan - Switches to panning if rectangular zoom is activated.
5. Reset - Restores the Maps to the default view.

`typescript

```
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
```

template:

```
`<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
<e-layers>
<e-layer [shapeData] = 'shapeData'></e-layer>
</e-layers>
</ejs-maps>`
})
```

```
export class AppComponent implements OnInit {
public zoomSettings: object;
public shapeData: object;
ngOnInit(): void {
this.zoomSettings = {
enable: true,
toolbarSettings: {
buttonSettings: {
toolbarItems: ['Zoom', 'ZoomIn', 'ZoomOut', 'Pan', 'Reset'],
}
}
};
this.shapeData = world_map;
}
}
`
```

[Pinch zooming](#)

To enable or disable the pinch zooming, use the [pinchZooming](#) property in [zoomSettings](#).

```
`typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
selector: 'app-container',
template:
```

```

`<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
<e-layers>
<e-layer [shapeData] = 'shapeData'></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
public zoomSettings: object;
public shapeData: object;
ngOnInit(): void {
this.zoomSettings = {
enable: true,
pinchZooming:true
};
this.shapeData = world_map;
}
}
`

```

Single-click zooming

To enable or disable the single-click zooming, use the [zoomOnClick](#) property in [zoomSettings](#).

```

` typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
selector: 'app-container',
template:
`<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
<e-layers>
<e-layer [shapeData] = 'shapeData'></e-layer>
</e-layers>
</ejs-maps>`

```



```

})
export class AppComponent implements OnInit {
  public zoomSettings: object;
  public shapeData: object;
  ngOnInit(): void {
    this.zoomSettings = {
      enable: true,
      zoomOnClick:true
    };
    this.shapeData = world_map;
  }
}
`

```

Double-click zooming

To enable or disable the double-click zooming, use the [doubleClickZoom](#) property in [zoomSettings](#).

```

` typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `

```

```

this.zoomSettings = {
  enable: true,
  doubleClickZoom:true
};
this.shapeData = world_map;
}
}
`

```

Mouse wheel zooming

To enable or disable mouse wheel zooming, use the [mouseWheelZoom](#) property in [zoomSettings](#).

```

` typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `

```

```

}
}
`

```

Selection zooming

To enable or disable selection zooming, use the [enableSelectionZooming](#) property in [zoomSettings](#). The [enablePanning](#) property must be set to **false** to enable the selection zooming in Maps.

```

` typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `

```

Setting minimum and maximum values for zoom factor

The zooming range can be adjusted using the [minZoom](#) and [maxZoom](#) properties in [zoomSettings](#). The minZoom value is set to 1 by default, and the maxZoom value is set to 10.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `

```

Zooming with animation

To zoom in or zoom out the Maps with animation, use the [animationDuration](#) property in [layers](#).

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

```

import { Maps, Zoom } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Zoom);
@Component({
  selector: 'app-container',
  template:
    `

```

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],

```

```

providers: [ZoomService],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData'></e-layer>
      </e-layers>
    </ejs-maps>`
  ))
export class AppComponent implements OnInit {
  public zoomSettings?: object;
  public shapeData?: object;
  ngOnInit(): void {
    this.zoomSettings = {
      enable: true,
    };
    this.shapeData = world_map;
  }
}

```

MAIN.TS

```

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

```

Customizing the zoom toolbar

The zoom toolbar can be customized by using the [toolbarSettings](#) option in the [zoomSettings](#). The following properties can be used to customize the zoom toolbar.

- [backgroundColor](#) - It is used to customize the background color of the zoom toolbar.
- [borderOpacity](#) - It is used to customize the opacity of the border of the zoom toolbar.
- [borderWidth](#) - It is used to customize the thickness of the border of the zoom toolbar.
- [borderColor](#) - It is used to customize the color of the border of the zoom toolbar.
- [horizontalAlignment](#) - It is used to position the zoom toolbar in near, far, and center positions to customize its horizontal placement.
- [verticalAlignment](#) - It is used to position the zoom toolbar in near, far, and center positions to customize its vertical placement.
- [orientation](#) - It is used to change the orientation (horizontal/vertical) of the zoom toolbar.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule

```

```

    ],
    providers: [ZoomService],
    standalone: true,
    selector: 'app-container',
    template:
      `

```

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 -->

Customizing the buttons in the zoom toolbar

The appearance of the buttons in the zoom toolbar can be customized by using the [buttonSettings](#) option in the [toolbarSettings](#) of the [zoomSettings](#) property. The following properties can be used to customize the zoom toolbar buttons.

- [fill](#) - It is used to set the background color of the buttons.
- [color](#) - It is used to customize the color of the icons inside the button.

- [borderOpacity](#) - It is used to set the opacity of the border of the zoom toolbar buttons.
- [borderWidth](#) - It is used to set the thickness of the border of the zoom toolbar buttons.
- [borderColor](#) - It is used to set the color of the border of the zoom toolbar buttons.
- [radius](#) - It is used to set the size of the button.
- [selectionColor](#) - It is used to set the color of the icons inside the button when selection is performed.
- [highlightColor](#) - It is used to change the color of the button when the mouse is hovered over it.
- [padding](#) - It is used to change the padding space between each button.
- [opacity](#) - It is used to change the opacity of the button.
- [toolbarItems](#) - It is used to change the items that should be displayed in the zoom toolbar. By default, zoom-in, zoom-out, and reset buttons will be available. Other options include selection zoom and panning.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ZoomService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData' [shapeSettings]="shapeSettings"></e-
layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public zoomSettings?: object;
  public shapeSettings?: object;
  public shapeData?: object;
  ngOnInit(): void {
    this.zoomSettings = {
      enable: true,
      toolbarSettings:{
        buttonSettings: {
          fill:'pink',
          padding: 10,
          toolbarItems: ['Zoom', 'ZoomIn', 'ZoomOut', 'Pan',
'Reset'],
          color: 'red',
          borderColor:'green',
          radius:35,
          selectionColor:'#d55e5e',
          highlightColor:'#5ed59a',
```



```

        opacity:0.6,
        borderWidth: 2
    }
    }
};
this.shapeSettings = {
    fill: '#C1DFF5'
}
this.shapeData = world_map;
}
}

```

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 -->

Customizing the tooltip of the zoom toolbar

The appearance of the tooltip of the zoom toolbar can be customized by using the [tooltipSettings](#) option in the [toolbarSettings](#) of the [zoomSettings](#) property. The following properties are available to customize the zoom toolbar tooltip.

- [visible](#) - Enables or disables the tooltip of the zoom toolbar.
- [fill](#) - It is used to change the background color of the tooltip of the zoom toolbar.
- [borderOpacity](#) - It is used to change the opacity of the border of the zoom toolbar's tooltip.
- [borderWidth](#) - It is used to change the thickness of the border of the zoom toolbar's tooltip.
- [borderColor](#) - It is used to change the color of the border of the zoom toolbar's tooltip.
- [fontColor](#) - It is used to change the color of the text in the tooltip of the zoom toolbar.
- [fontFamily](#) - It is used to change the font family of the text in the tooltip of the zoom toolbar.
- [fontStyle](#) - It is used to change the font style of the text in the tooltip of the zoom toolbar.
- [fontWeight](#) - It is used to change the font weight of the text in the tooltip of the zoom toolbar.
- [fontSize](#) - It is used to change the size of the text in the tooltip of the zoom toolbar.
- [fontOpacity](#) - It is used to change the opacity of the text in the tooltip of the zoom toolbar.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ZoomService],
  standalone: true,

```

```

    selector: 'app-container',
    template:
    `<ejs-maps id='rn-container' [zoomSettings] = 'zoomSettings'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData' [shapeSettings]="shapeSettings"></e-
layer>
      </e-layers>
    </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public zoomSettings?: object;
    public shapeSettings?: object;
    public shapeData?: object;
    ngOnInit(): void {
      this.zoomSettings = {
        enable: true,
        toolbarSettings: {
          tooltipSettings: {
            visible: true,
            borderWidth: 2,
            borderColor: 'green',
            fontColor: 'black',
            fill: 'violet',
            fontFamily: 'Times New Roman',
            fontWeight: 200,
            fontSize: '22px',
            fontOpacity: 1
          },
          buttonSettings: {
            toolbarItems: ['Zoom', 'ZoomIn', 'ZoomOut', 'Pan',
'Reset']
          }
        }
      };
      this.shapeSettings = {
        fill: '#C1DFF5'
      }
      this.shapeData = world_map;
    }
  }
}

```

MAIN.TS

```

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

```

Selection

Each shape in the Maps can be selected and deselected during interaction with the shapes. Selection is enabled by setting the [enable](#) property of [selectionSettings](#) to **true**.

The following properties are available to customize the selection of Maps elements such as shapes, bubbles, markers and legends.

- [border](#) - To customize the color, width and opacity of the border of which element is selected in Maps.
- [fill](#) - Applies the color for the element that is selected.
- [opacity](#) - To customize the transparency for the element that is selected.
- [enableMultiSelect](#) - To enable or disable the selection for multiple shapes or markers or bubbles in the Maps.

By tapping on the specific legend, the shapes which are bounded to the selected legend is also selected and vice versa.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, SelectionService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [legendSettings]='legendSettings'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData'
          [shapePropertyPath]='shapePropertyPath' [shapeDataPath]='shapeDataPath'
          [dataSource]='dataSource' [shapeSettings]='shapeSettings'
          [selectionSettings] = 'selectionSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public selectionSettings?: object;
  public shapeData?: object;
  public shapePropertyPath?: string;
  public shapeDataPath?: string;
  public dataSource?: object;
  public shapeSettings?: object;
  public legendSettings?: object;
  ngOnInit(): void {
    this.selectionSettings = {
      enable: true,
      fill: 'blue',
      border: { color: 'white', width: 2 }
    };
    this.shapeData = world_map;
    this.shapePropertyPath = "name";
    this.shapeDataPath = "Country";
    this.dataSource = [
      { "Country": "China", "Membership": "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
    ],
  }
}
```

```

        { "Country": "Russia", "Membership": "Permanent" },
        { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
        { "Country": "Poland", "Membership": "Non-Permanent" },
        { "Country": "Sweden", "Membership": "Non-Permanent" }
    ];
    this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
            {
                value: 'Permanent', color: '#D84444'
            },
            {
                value: 'Non-Permanent', color: '#316DB5'
            }
        ]
    };
    this.legendSettings = {
        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));

```

Enable selection for bubbles

To enable the selection for bubbles in Maps, set the [selectionSettings](#) in [bubbleSettings](#) and set the [enable](#) property of [selectionSettings](#) as **true**.

To use the bubble feature, the Bubble module must be injected.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { BubbleService, SelectionService } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
    imports: [
        MapsModule
    ],
    providers: [BubbleService, SelectionService],
    standalone: true,
    selector: 'app-container',
    template:
        `<ejs-maps id='rn-container'>
        <e-layers>
        <e-layer [shapeData]= 'shapeData' [bubbleSettings] =
        'bubbleSettings'></e-layer>
        </e-layers>
        `
})

```

```

    </ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public shapeData?: object;
    public shapeDataPath?: object | any;
    public shapePropertyPath?: object | any;
    public bubbleSettings?: object;
    ngOnInit(): void {
      this.shapeData = world_map;
      this.shapeDataPath = 'name',
      this.shapePropertyPath = 'name',
      this.bubbleSettings = [{
        visible: true,
        dataSource: [
          { name: 'India', population: '38332521' },
          { name: 'South Africa', population: '19651127' },
          { name: 'Pakistan', population: '3090416' }
        ],
        selectionSettings: {
          enable: true,
          fill: 'green',
          border: { color: 'white', width: 2 }
        },
        valuePath: 'population'
      }]
    }
  }
}

```

MAIN.TS

```

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

```

Enable selection for markers

To enable the selection for markers in Maps, set the [selectionSettings](#) in the [markerSettings](#) and set the [enable](#) property of the [selectionSettings](#) as **true**.

To use the marker feature, the Marker module must be injected.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, SelectionService],
})

```

```

standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
  })
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      height: 20,
      width: 20,
      fill: 'green',
      shape: 'Balloon',
      selectionSettings: {
        enable: true,
        fill: 'blue',
        border: { color: 'white', width: 2 }
      },
      dataSource: [
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name: 'Europe' },
        { latitude: 59.88893689676585, longitude: -109.3359375,
name: 'North America' },
        { latitude: -6.64607562172573, longitude: -
55.546874999999999, name: 'South America' }
      ]
    }];
  }
}

```

MAIN.TS

```

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

```

Enable selection for polygons

When the [enable](#) property of [selectionSettings](#) is set to **true**, the polygon shapes can be selected via user interaction. The following properties are available in [selectionSettings](#) to customize the polygon shape when it is selected.

- [enableMultiSelect](#) - It is used to enable multiple selection of polygon shapes.
- [fill](#) - It is used to change the color of the selected polygon shape.
- [opacity](#) - It is used to change the opacity of the selected polygon shape.

- [border](#) - This property is used to change the color, width, and opacity of the border of the selected polygon shape.

To use the polygon feature, the **Polygon** module must be injected, as described in [this link](#).

The following example shows how to select the polygon shape in the geometry map.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule, PolygonService, SelectionService, HighlightService }
from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PolygonService, SelectionService, HighlightService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="control-section">
<div align="center">
<ejs-maps id='container' style="display:block;">
<e-layers>
<e-layer [shapeData]='shapeData' [polygonSettings]='polygonSettings'></e-
layer>
</e-layers>
</ejs-maps>
</div>
</div>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public polygonSettings?: object;
  ngOnInit(): void {
    this.polygonSettings = {
      polygons: [
        {
          points: [
            { longitude: -1.8920678947185365, latitude: 35.06195799239681 },
            { longitude: -1.6479633699113947, latitude: 33.58989612266137 },
            { longitude: -1.4201220366858252, latitude: 32.819439646045254 },
            { longitude: -1.197974596225663, latitude: 32.26940895444655 },
            { longitude: -2.891112397949655, latitude: 32.10303058820031 },
            { longitude: -3.8246984550501963, latitude: 31.34551662687602 },
            { longitude: -3.720166273688733, latitude: 30.758086682848685 },
            { longitude: -5.6571886081189575, latitude: 29.613582597203006 },
            { longitude: -7.423353242214745, latitude: 29.44328441403087 },
            { longitude: -8.6048931685323, latitude: 28.761444633616776 },
            { longitude: -8.695726975465703, latitude: 27.353491085576195 },
            { longitude: 3.837867279970908, latitude: 19.15916564839422 },
            { longitude: 6.0705408799045415, latitude: 19.48749097192868 },
          ]
        }
      ]
    }
  }
}
```

```

        { longitude: 12.055736352807713, latitude: 23.694596786078293 },
        { longitude: 11.272522332402986, latitude: 24.289329186946034 },
        { longitude: 10.30872578261932, latitude: 24.65419958524693 },
        { longitude: 9.910236690050027, latitude: 25.48943950947175 },
        { longitude: 9.432639882414293, latitude: 26.398372489836902 },
        { longitude: 9.898266456582292, latitude: 26.73489453809293 },
        { longitude: 9.560243026853641, latitude: 30.31040379467153 },
        { longitude: 8.943853847283322, latitude: 32.350324876652195 },
        { longitude: 7.57004059025715, latitude: 33.75071049019398 },
        { longitude: 8.0906322609153, latitude: 34.69043151009983 },
        { longitude: 8.363285449347273, latitude: 35.38654406371319 },
        { longitude: 8.26139549449448, latitude: 36.44751078733985 },
        { longitude: 8.61100824823302, latitude: 36.881913362940196 },
        { longitude: 7.4216488925819135, latitude: 37.021408008916254 },
        { longitude: 6.461182254165351, latitude: 36.99092409199429 },
        { longitude: 5.297178918070159, latitude: 36.69985479014656 },
        { longitude: 3.6718056161224695, latitude: 36.86470546831693 },
        { longitude: 1.2050052555659931, latitude: 36.57658056301722 },
        { longitude: -0.26968570003779746, latitude: 35.806903541813625
    },
    { longitude: -0.995191786435754, latitude: 35.58466127904214 },
    { longitude: -1.8920678947185365, latitude: 35.06195799239681 }
  ],
  fill: 'blue',
  opacity: 0.7,
  borderColor: 'green',
  borderWidth: 2,
  borderOpacity: 0.7
},
highlightSettings: {
  enable: true,
  fill: 'blue',
  opacity: 0.7,
  border: {
    color: 'green',
    width: 2,
    opacity: 0.7
  }
},
selectionSettings: {
  enable: true,
  fill: 'violet',
  enableMultiSelect: false,
  opacity: 0.8,
  border: {
    color: 'cyan',
    opacity: 1,
    width: 7
  }
}
};
this.shapeData = world_map;
}
}

```


MAIN.TS

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

Public method for the shape selection

The [shapeSelection](#) method can be used to select each shape in the Maps.

LayerIndex, propertyName, country name, and selected value as a boolean state(true / false) are the input parameters for this method.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' #maps>
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [selectionSettings]
    ='selectionSettings'></e-layer>
  </e-layers>
</ejs-maps> <button id='select' (click)='select()'>select</button>
<button id='unselect' (click)='unselect()'>unselect</button>`
})
export class AppComponent {
  @ViewChild('maps')
  public mapObj?: MapsComponent;
  public shapeData: object = world_map;
  public selectionSettings: object = {
    enable: true,
    fill: 'green',
    border: { color: 'white', width: 2 }
  };
  select() {
    this.mapObj?.shapeSelection(0, "continent", "Asia", true);
  };
  unselect() {
    this.mapObj?.shapeSelection(0, "continent", "Asia", false);
  }
}
```

MAIN.TS

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

Initial shape selection

The shape is initially selected using the [initialShapeSelection](#), and the values are mapped to the [shapePath](#) and [shapeValue](#).

initialShapeSelection is an Array property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [SelectionService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

MAIN.TS

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

Initial marker selection

Using the [initialMarkerSelection](#), the marker shape can be selected initially. Markers render based on the [latitude](#) and [longitude](#) values.

Note: initialMarkerSelection is an Array property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, SelectionService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
    'markerSettings'></e-layer>
    </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      height: 20,
      width: 20,
      fill: 'green',
      shape: 'Balloon',
      initialMarkerSelection: [{
        latitude: -6.64607562172573, longitude: -55.54687499999999
      }],
      selectionSettings: {
        enable: true,
        fill: 'blue',
        opacity: 1,
        border: { color: 'white', width: 2, opacity: 1 }
      },
    ],
    dataSource: [
```

```

        { latitude: 49.95121990866204, longitude:
18.468749999999998, name: 'Europe' },
        { latitude: 59.88893689676585, longitude: -109.3359375,
name: 'North America'},
        { latitude: -6.64607562172573, longitude: -
55.546874999999999, name: 'South America'}
    ]
  }];
}
}

```

MAIN.TS

```

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

```

Highlight

Each shape in the Maps can be highlighted during mouse hover on the Maps elements such as shapes, bubbles, markers and legends. Highlight is enabled by setting the [enable](#) property of [highlightSettings](#) to **true**.

The following properties are available to customize the highlight of Maps elements such as shapes, bubbles, markers and legends.

- [border](#) - To customize the color, width and opacity of the border of which element is highlighted in Maps.
- [fill](#) - Applies the color for the element that is highlighted.
- [opacity](#) - To customize the transparency for the element that is highlighted.

Hovering on the specific legend, the shapes which are bounded to the selected legend is also highlighted and vice versa.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, HighlightService } from '@syncfusion/ej2-angular-
maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, HighlightService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' [legendSettings]='legendSettings'>
  <e-layers>

```

```

    <e-layer [shapeData] = 'shapeData'
    [shapePropertyPath]='shapePropertyPath' [shapeDataPath]='shapeDataPath'
    [dataSource]='dataSource' [shapeSettings]='shapeSettings'
    [highlightSettings] ='highlightSettings'></e-layer>
  </e-layers>
</ejs-maps>`
  })
  export class AppComponent implements OnInit {
    public highlightSettings?: object;
    public shapeData?: object;
    public shapePropertyPath?: string;
    public shapeDataPath?: string;
    public dataSource?: object;
    public shapeSettings?: object;
    public legendSettings?: object;
    ngOnInit(): void {
      this.highlightSettings = {
        enable: true,
        fill: 'green',
        border: { color: 'white', width: 2}
      }
      this.shapeData = world_map;
      this.shapePropertyPath= "name";
      this.shapeDataPath = "Country";
      this.dataSource = [
        { "Country": "China", "Membership": "Permanent"},
        { "Country": "France", "Membership": "Permanent" },
        { "Country": "Russia", "Membership": "Permanent"},
        { "Country": "Kazakhstan", "Membership": "Non-Permanent"},
        { "Country": "Poland", "Membership": "Non-Permanent"},
        { "Country": "Sweden", "Membership": "Non-Permanent"}
      ];
      this.shapeSettings = {
        colorValuePath: 'Membership',
        colorMapping: [
          {
            value: 'Permanent', color: '#D84444'
          },
          {
            value: 'Non-Permanent', color: '#316DB5'
          }
        ]
      };
      this.legendSettings= {
        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));

```

Enable highlight for bubbles

To enable the highlight for bubbles in Maps, set the [highlightSettings](#) in [bubbleSettings](#) and set the [enable](#) property of [highlightSettings](#) as **true**.

To use the bubble feature, the Bubble module must be injected.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { BubbleService, HighlightService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [BubbleService, HighlightService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

MAIN.TS

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

Enable highlight for markers

To enable the highlight for markers in Maps, set the [highlightSettings](#) in [markerSettings](#) and set the [enable](#) property of [highlightSettings](#) as **true**.

Note: To use the marker feature, the Marker module must be injected.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { MarkerService, HighlightService } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, HighlightService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container'>
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [markerSettings] =
'markerSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public markerSettings?: object;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.markerSettings = [{
      visible: true,
      height: 20,
      width: 20,
      fill: 'green',
      shape: 'Balloon',
      highlightSettings: {
        enable: true,
        fill: 'blue',
        border: { color: 'white', width: 2 }
      },
      dataSource: [
        { latitude: 49.95121990866204, longitude:
18.468749999999998, name: 'Europe' },
```

```

        { latitude: 59.88893689676585, longitude: -109.3359375,
name: 'North America'},
        { latitude: -6.64607562172573, longitude: -
55.54687499999999, name: 'South America'}
    ]
  }];
}
}

```

MAIN.TS

```

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

```

Enable highlight for polygons

The polygon shapes can be highlighted via user interaction if the [enable](#) property of [highlightSettings](#) is set to **true**. The following properties are available in [highlightSettings](#) to customize the polygon shape when it is highlighted.

- [fill](#) - It is used to change the color of the highlighted polygon shape.
- [opacity](#) - It is used to change the opacity of the highlighted polygon shape.
- [border](#) - This property is used to change the color, width, and opacity of the border of the highlighted polygon shape.

To use the polygon feature, the **Polygon** module must be injected, as described in [this link](#).

The following example shows how to highlight a polygon shape on a geometry map.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule, PolygonService, SelectionService, HighlightService }
from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PolygonService, SelectionService, HighlightService],
  standalone: true,
  selector: 'app-container',
  template: `<div class="control-section">
<div align="center">
<ejs-maps id="container" style="display:block;">
<e-layers>
<e-layer [shapeData]='shapeData' [polygonSettings]='polygonSettings'></e-
layer>
</e-layers>
</ejs-maps>
</div>
`
})

```



```

</div>`,
  encapsulation: ViewEncapsulation.None,
})
export class AppComponent implements OnInit {
  public shapeData?: object;
  public polygonSettings?: object;
  ngOnInit(): void {
    this.polygonSettings = {
      polygons: [
        {
          points: [
            { longitude: -1.8920678947185365, latitude: 35.06195799239681 },
            { longitude: -1.6479633699113947, latitude: 33.58989612266137 },
            { longitude: -1.4201220366858252, latitude: 32.819439646045254 },
          ],
          { longitude: -1.197974596225663, latitude: 32.26940895444655 },
          { longitude: -2.891112397949655, latitude: 32.10303058820031 },
          { longitude: -3.8246984550501963, latitude: 31.34551662687602 },
          { longitude: -3.720166273688733, latitude: 30.758086682848685 },
          { longitude: -5.6571886081189575, latitude: 29.613582597203006 },
        },
        { longitude: -7.423353242214745, latitude: 29.44328441403087 },
        { longitude: -8.6048931685323, latitude: 28.761444633616776 },
        { longitude: -8.695726975465703, latitude: 27.353491085576195 },
        { longitude: 3.837867279970908, latitude: 19.15916564839422 },
        { longitude: 6.0705408799045415, latitude: 19.48749097192868 },
        { longitude: 12.055736352807713, latitude: 23.694596786078293 },
        { longitude: 11.272522332402986, latitude: 24.289329186946034 },
        { longitude: 10.30872578261932, latitude: 24.65419958524693 },
        { longitude: 9.910236690050027, latitude: 25.48943950947175 },
        { longitude: 9.432639882414293, latitude: 26.398372489836902 },
        { longitude: 9.898266456582292, latitude: 26.73489453809293 },
        { longitude: 9.560243026853641, latitude: 30.31040379467153 },
        { longitude: 8.943853847283322, latitude: 32.350324876652195 },
        { longitude: 7.57004059025715, latitude: 33.75071049019398 },
        { longitude: 8.0906322609153, latitude: 34.69043151009983 },
        { longitude: 8.363285449347273, latitude: 35.38654406371319 },
        { longitude: 8.26139549449448, latitude: 36.44751078733985 },
        { longitude: 8.61100824823302, latitude: 36.881913362940196 },
        { longitude: 7.4216488925819135, latitude: 37.021408008916254 },
        { longitude: 6.461182254165351, latitude: 36.99092409199429 },
        { longitude: 5.297178918070159, latitude: 36.69985479014656 },
        { longitude: 3.6718056161224695, latitude: 36.86470546831693 },
        { longitude: 1.2050052555659931, latitude: 36.57658056301722 },
        { longitude: -0.26968570003779746, latitude: 35.806903541813625 },
      ],
      { longitude: -0.995191786435754, latitude: 35.58466127904214 },
      { longitude: -1.8920678947185365, latitude: 35.06195799239681 }
    ],
    fill: 'red',
    opacity: 0.7,
    borderColor: 'green',
    borderWidth: 2,
    borderOpacity: 0.7
  }
},
highlightSettings: {

```

```

        enable: true,
        fill: 'yellow',
        opacity: 0.4,
        border: {
            color: 'blue',
            opacity: 0.6,
            width: 4
        }
    },
    selectionSettings: {
        enable: true,
        fill: 'red',
        opacity: 0.7,
        border: {
            color: 'green',
            width: 2,
            opacity: 0.7
        }
    }
};
this.shapeData = world_map;
}
}

```

MAIN.TS

```

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

```

Tooltip

On mouse over or touch end event, the tooltip is used to get more information about the layer, bubble, or marker. To enable tooltip in Maps, the **Tooltip** module must be injected into Maps using **Maps.Inject(Tooltip)** method. It can be enabled separately for layer or bubble or marker by using the [visible](#) property of [tooltipSettings](#) as **true**. The [valuePath](#) property in the tooltip takes the field name that presents in data source and displays that value as tooltip text. The [tooltipDisplayMode](#) property is used to change the display mode of the tooltip in Maps. Following display modes of tooltip are available in the Maps component. By default, [tooltipDisplayMode](#) is set to **MouseMove**.

- MouseMove
- Click
- DoubleClick

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { MapsTooltipService } from '@syncfusion/ej2-angular-maps';
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({

```

```

imports: [
    MapsModule
],
providers: [MapsTooltipService],
standalone: true,
selector: 'app-container',
template:
`<ejs-maps id='rn-container'>
  <e-layers>
    <e-layer [shapeData] = 'shapeData' [tooltipSettings]
='tooltipSettings'></e-layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public tooltipSettings?: object;
  public shapeData?: object;
  ngOnInit(): void {
    this.tooltipSettings = {
      visible: true,
      valuePath: 'name'
    }
    this.shapeData = world_map;
  }
}

```

MAIN.TS

```

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

```

app.module.ts

Injecting MapsTooltipService into NgModule.

`typescript

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the MapsModule for the Maps component
import { MapsModule, MapsTooltipService } from '@syncfusion/ej2-angular-maps';
import { AppComponent } from './app.component';
@NgModule({
  // declaration of ej2-angular-maps module into NgModule
  imports: [ BrowserModule, MapsModule ],
  declarations: [ AppComponent ],
  providers: [MapsTooltipService],

```

```
bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Customization

The following properties are available to customize the tooltip of the Maps component.

- [border](#) - To customize the color, width and opacity of the border of the tooltip in layers, markers, and bubbles of Maps.
- [fill](#) - Applies the color of the tooltip in layers, markers, and bubbles of Maps.
- [format](#) - To customize the format of the tooltip in layers, markers, and bubbles of Maps
- [textStyle](#) - To customize the style of the text in the tooltip for layers, markers, and bubbles of Maps.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container'>
      <e-layers>
        <e-layer [shapeData] = 'shapeData' [tooltipSettings]
        ='tooltipSettings'></e-layer>
      </e-layers>
    </ejs-maps>`
})
export class AppComponent implements OnInit {
  public tooltipSettings?: object;
  public shapeData?: object;
  ngOnInit(): void {
    this.tooltipSettings = {
      visible: true,
      valuePath: 'name',
      fill: '#D0D0D0',
      textStyle: {
        color: 'green',
        fontFamily: 'Times New Roman',
        fontStyle: 'Sans-serif'
      }
    }
    this.shapeData = world_map;
  }
}
```

```
}
}
```

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

The HTML element can be rendered in the tooltip of the Maps using the [template](#) property of the [tooltipSettings](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService } from '@syncfusion/ej2-angular-maps'
import { Component } from '@angular/core';
import { ShapeSettings } from '@syncfusion/ej2-angular-maps';
import { world_map } from './world-map';
import { default_data } from './data';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-maps id='rn-container'>
    <e-layers>
      <e-layer [shapeData] ='shapeData' [shapePropertyPath]='shapePropertyPath'
[shapeDataPath]='shapeDataPath' [dataSource]='dataSource'
[shapeSettings]='shapeSettings' [tooltipSettings] ='tooltipSettings'></e-
layer>
    </e-layers>
  </ejs-maps>`
})
export class AppComponent {
  public tooltipSettings: object = {
    visible: true,
    valuePath: 'continent',
    template: '<div style="width:60px; text-align:center; background-
color: white; border: 2px solid black; padding-bottom: 10px;padding-top:
10px;padding-left: 10px;padding-right:
10px;"><span>${continent}</span></div>',
    textStyle: {
      color: 'black'
    }
  }
  public shapePropertyPath: string = "continent";
  public shapeDataPath: string = "continent";
  public dataSource: object = default_data;
```

```

    public shapeData: object = world_map;
    public shapeSettings?: ShapeSettings;
}

```

MAIN.TS

```

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

```

Print in Angular Maps component

Print

The rendered Maps can be printed directly from the browser by calling the [print](#) method. To use the print functionality, the **PrintService** must be injected into the Maps using **providers** of the Angular component and set the [allowPrint](#) property to **true**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, PrintService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewChild } from '@angular/core';
import { world_map } from './world-map';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, PrintService],
  standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' #maps [allowPrint]=true [legendSettings] =
'legendSettings'>
    <e-layers>
      <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings]= 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps> <button id='print' (click)='print()'>Print</button>`
})
export class AppComponent {
  @ViewChild('maps')
  public mapObj?: MapsComponent;
  public dataSource: object[] = [
    { "Country": "China", "Membership": "Permanent" },
    { "Country": "France", "Membership": "Permanent" },
    { "Country": "Russia", "Membership": "Permanent" },
    { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
    { "Country": "Poland", "Membership": "Non-Permanent" },
    { "Country": "Sweden", "Membership": "Non-Permanent" }];
  public shapeData: object = world_map;
  public shapePropertyPath: string = 'name';
}

```

```

public shapeDataPath: string= 'Country';
public shapeSettings: object = {
  colorValuePath: 'Membership',
  colorMapping: [
    {
      value: 'Permanent', color: '#D84444'
    },
    {
      value: 'Non-Permanent', color: '#316DB5'
    }
  ]
};
public legendSettings: object = {
  visible: true
};
public print() {
  this.mapObj?.print();
};
}

```

MAIN.TS

```

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

```

Export

Image Export

To use the image export functionality in Maps, **ImageExport** module must be injected into the Maps using **Maps.Inject(ImageExport)** method and set the [allowImageExport](#) property to **true**. The rendered Maps can be exported as an image using the [export](#) method. The method requires two parameters: image type and file name. The Maps can be exported as an image in the following formats.

- JPEG
- PNG
- SVG

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { LegendService, ImageExportService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewChild } from '@angular/core';
import { world_map } from './world-map';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
  imports: [
    MapsModule
  ],
  providers: [LegendService, ImageExportService],
  standalone: true,

```

```

    selector: 'app-container',
    template:
    `<ejs-maps id='rn-container' #maps [allowImageExport]=true
[legendSettings] = 'legendSettings'>
    <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
    </e-layers>
    </ejs-maps> <button id='print' (click)='export()'>Export</button>`
  })
  export class AppComponent {
    @ViewChild('maps')
    public mapObj?: MapsComponent;
    public dataSource: object[] = [
      { "Country": "China", "Membership": "Permanent" },
      { "Country": "France", "Membership": "Permanent" },
      { "Country": "Russia", "Membership": "Permanent" },
      { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
      { "Country": "Poland", "Membership": "Non-Permanent" },
      { "Country": "Sweden", "Membership": "Non-Permanent" }];
    public shapeData: object = world_map;
    public shapePropertyPath: string = 'name';
    public shapeDataPath: string = 'Country';
    public shapeSettings: object = {
      colorValuePath: 'Membership',
      colorMapping: [
        {
          value: 'Permanent', color: '#D84444'
        },
        {
          value: 'Non-Permanent', color: '#316DB5'
        }
      ]
    };
    public legendSettings: object = {
      visible: true
    };
    public export() {
      this.mapObj?.export('JPEG', 'Maps');
    }
  }
}

```

MAIN.TS

```

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

```

Exporting Maps as base64 string of the file

We can get the image file as base64 string for the JPEG and PNG formats. The rendered Maps can be exported to image as a base64 string using the [export](#) method. There are four parameters required: image type, file name, orientation of the exported PDF document which must be set as **null** for image export and finally **allowDownload** which should be set as **false** to return base64 string.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ImageExportService, LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { world_map } from './world-map';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ImageExportService, LegendService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```

```

        promise.then((data)=>{
            (document.getElementById('data') as HTMLElement).innerHTML =
data;
        })
    };
}

```

MAIN.TS

```

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

```

PDF Export

To use the PDF export functionality, **PdfExport** module must be injected into the Maps using **Maps.Inject(PdfExport)** method and set the [allowPdfExport](#) property to **true**. The rendered map can be exported as PDF using the [export](#) method. The [export](#) method requires three parameters: file type, file name and orientation of the PDF document. The orientation setting is optional and **0** indicates portrait and **1** indicates landscape.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { PdfExportService, LegendService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { world_map } from './world-map';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
  imports: [
    MapsModule
  ],
  providers: [PdfExportService, LegendService],
  standalone: true,
  selector: 'app-container',
  template:
`<ejs-maps id='rn-container' #maps [allowPdfExport]=true
[legendSettings] = 'legendSettings'>
  <e-layers>
    <e-layer [shapeData]= 'shapeData' [shapePropertyPath]=
'shapePropertyPath' [shapeDataPath]= 'shapeDataPath' [dataSource] =
'dataSource' [shapeSettings] = 'shapeSettings'></e-layer>
  </e-layers>
</ejs-maps> <button id='print' (click)= 'export()' >Export</button>`
})
export class AppComponent {
  @ViewChild('maps')
  public mapObj?: MapsComponent;
  public dataSource: object[] = [
    { "Country": "China", "Membership": "Permanent" },
    { "Country": "France", "Membership": "Permanent" },
    { "Country": "Russia", "Membership": "Permanent" },

```

```

        { "Country": "Kazakhstan", "Membership": "Non-Permanent" },
        { "Country": "Poland", "Membership": "Non-Permanent" },
        { "Country": "Sweden", "Membership": "Non-Permanent" } ] ];
    public shapeData: object = world_map;
    public shapePropertyPath: string = 'name';
    public shapeDataPath: string = 'Country';
    public shapeSettings: object = {
        colorValuePath: 'Membership',
        colorMapping: [
            {
                value: 'Permanent', color: '#D84444'
            },
            {
                value: 'Non-Permanent', color: '#316DB5'
            }
        ]
    };
    public legendSettings: object = {
        visible: true
    };
    public export() {
        this.mapObj?.export('PDF', 'Maps', 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 exporting of the map as base64 string is not supported in the PDF export.

Export the tile maps

The rendered Maps with providers such as OSM, Bing and Google static maps can be exported using the [export](#) method. It supports the following export formats.

- JPEG
- PNG
- PDF

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { PdfExportService, ImageExportService, LegendService } from '@syncfusion/ej2-angular-maps';
import { Component, ViewChild } from '@angular/core';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
    imports: [
        MapsModule
    ],

```

```

providers: [PdfExportService, ImageExportService, LegendService],
standalone: true,
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' #maps [allowPdfExport]=true
[allowImageExport]=true [titleSettings] = 'titleSettings'>
    <e-layers>
    <e-layer [urlTemplate]= 'urlTemplate'></e-layer>
    </e-layers>
    </ejs-maps> <button id='export' (click)='export()'>Export</button>`
  })
export class AppComponent {
  @ViewChild('maps')
  public mapObj?: MapsComponent;
  public urlTemplate =
'https://tile.openstreetmap.org/level/tileX/tileY.png';
  public titleSettings: object = {
    text: 'OSM'
  };
  public export() {
    this.mapObj?.export('JPEG', 'Maps');
  }
}

```

MAIN.TS

```

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

```

State persistence in Angular Maps component

State Persistence

State persistence allows the Maps to retain the current model value in the browser cookies for state maintenance. This action is handled through the [enablePersistence](#) property which is set to **false** by default. When this property is set to **true**, some of the Maps component model values are preserved even after the page is refreshed.

`typescript

```

import { Component, OnInit } from '@angular/core';
import { Maps, Selection } from '@syncfusion/ej2-angular-maps';
import { world_map } from 'world-map.ts';
Maps.Inject(Selection);
@Component({
  selector: 'app-container',
  template:
    `<ejs-maps id='rn-container' [enablePersistence] = 'enablePersistence' [zoomSettings] = 'zoomSettings'>
    <e-layers>

```

```

<e-layer [shapeData]= 'shapeData'></e-layer>
</e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public shapeData: object;
  public zoomSettings: object;
  public enablePersistence: boolean;
  ngOnInit(): void {
    this.shapeData = world_map;
    this.enablePersistence = true;
    this.zoomSettings = {
      enable: true,
    };
  }
}
`

```

Accessibility in Angular Maps component

The Maps 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 Maps component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2 Support](#) | |

| [Section 508 Support](#) | |

| [Screen Reader 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

To meet accessibility standards, the Maps component follows to the [WAI-ARIA](#) patterns. In the Maps component, the following ARIA attributes are used:

| Attributes | Purpose |

| --- | --- |

| **role=region** | It specifies the Maps areas that do not support interactive functions like selection and highlight. |

| **role=button** | It specifies the Maps areas where interactive functions such as selection and highlight are available. |

| **aria-label** | Provides an accessible name for Maps elements such as geometric map shapes, title, subtitle, legend title, legend item labels, data labels, and so on. To learn more, see the next topic. |

Screen reading in Maps

Accessibility in the Maps component ensures that all users, regardless of ability or disability, can use screen reading. The following Map elements will be read aloud using screen reading software, such as Narrator for Windows.

| Elements | Description |

| --- | --- |

| Shapes in the layer | Reads the names of the geographical shapes (such as countries, states, and regions) that appear on the Maps. |

| Title | Reads the title content in the Maps. |

| Subtitle | Reads the title below the main title content in the Maps. |

- | Legend title | Reads the contents of the legend's title as specified in Maps. |
- | Legend item label | Reads the label of a legend item in Maps. |
- | Data label | Reads the label specified for the shapes in the Maps layer. |
- | Annotation | Reads the content specified in the annotation. |
- | Marker template | Reads the content provided in the marker template. |
- | Tooltip template | Reads the content provided in the tooltip template. |
- | Data label template | Reads the content provided in the data label template. |

Keyboard Navigation

All the Maps actions can be controlled via keyboard keys. The applicable key combinations and their relative functionalities are listed below for the appropriate UI features available in the component.

Interaction Keys | Description

- Tab** | Moves to the next focusable element on the map, such as the legend or shape.
- Shift + Tab** | Moves to the previous focusable element on the map, such as the legend or shape.
- +** | When zooming is enabled, zoom in operation can be performed.
- | When zooming is enabled, zoom out operation can be performed.
- Left arrow** | When zoomed in, the map can be scrolled to the left.
- Right arrow** | When zoomed in, the map can be scrolled to the right.
- Up arrow** | When zoomed in, the map can be scrolled upward.
- Down arrow** | When zoomed in, the map can be scrolled downward.
- R** | When zooming is enabled, reset operation can be performed.
- Enter** | The page can be navigated to the next and previous states in legend. Similarly, the selection can be made while navigating over the shape.

Ensuring accessibility

The Maps component's accessibility levels are ensured using an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Maps component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Maps component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Internationalization in Angular Maps component

Maps provide support for internationalization for the below elements.

- Data label
- Tooltip

For more information about number and date formatter, refer to the [internationalization](#) section.

<!-- markdownlint-disable MD036 -->

Globalization

Globalization is the process of designing and developing a component that works in different cultures/locales. Internationalization library is used to globalize number, date, time values in Maps component using [format](#) property in the [Maps](#).

Numeric Format

The numeric formats such as currency, percentage and so on can be displayed in the tooltip and data labels of the Maps using the [format](#) property in the [Maps](#). In the below example, the tooltip is globalized to **German** culture. When setting the [useGroupingSeparator](#) property as **true**, the numeric text in the Maps separates with the comma separator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService],
  standalone: true,
  selector: 'app-container',
  template:
    `

```



```

    { "Country": "Poland", "Membership": "Non-Permanent",
population: '90332521'},
    { "Country": "Sweden", "Membership": "Non-Permanent",
population: '383521'}
  ];
  this.shapeData = world_map;
  this.shapePropertyPath = 'name';
  this.shapeDataPath = 'Country';
  this.shapeSettings = {
    colorValuePath: 'Membership',
    colorMapping: [
      {
        value: 'Permanent', color: '#D84444'
      },
      {
        value: 'Non-Permanent', color: '#316DB5'
      }
    ]
  };
  this.tooltipSettings = {
    visible: true,
    valuePath: 'population'
  };
}
}

```

MAIN.TS

```

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

```

Localization in Angular Maps component

The localization library allows localizing the default text content of the Maps component. The Maps component has the static text of some features such as tooltip of zoom toolbar, and that can be changed to any other culture(Arabic, Deutsch, French, etc) by defining the locale value and translation object.

<!-- markdownlint-disable MD033 -->

<!-- markdownlint-disable MD010 -->

Locale key words	Text to display
Zoom	Zoom
ZoomIn	Zoom In
ZoomOut	Zoom Out
Reset	Reset
Pan	Pan

To load translation object in the application, use `load` function of `L10n` class. For more information about localization, refer [here](#).

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MapsTooltipService, ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, OnInit } from '@angular/core';
import { world_map } from './world-map';
import { L10n } from '@syncfusion/ej2-base';
L10n.load({
  'ar-AR': {
    'maps': {
      ZoomIn: 'تكبير',
      ZoomOut: 'تصغير',
      Zoom: 'زوم',
      Pan: 'مقلاة',
      Reset: 'إعادة تعيين'
    }
  }
});
@Component({
  imports: [
    MapsModule
  ],
  providers: [MapsTooltipService, ZoomService],
  standalone: true,
  selector: 'app-container',
  template: `<ejs-maps id='container' [locale]="Locale"
[zoomSettings]='zoom'>
  <e-layers>
    <e-layer [shapeData]="mapData" [tooltipSettings]='tooltipSettings'></e-
layer>
  </e-layers>
</ejs-maps>`
})
export class AppComponent implements OnInit {
  public mapData: object[] = world_map as any;
  public Locale: string = 'ar-AR';
  public zoom: object = {
    enable: true
  };
  public tooltipSettings: object = {
    enable: true
  };
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

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

Methods in Angular Maps component

Methods

This section explains the methods used in the Maps control.

getMinMaxLatitudeLongitude

The `getMinMaxLatitudeLongitude` method returns the minimum and maximum latitude and longitude values of the Maps visible area. This method returns a [IMinMaxLatitudeLongitude](#) object that contains the Maps minimum and maximum latitude and longitude coordinates.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, ZoomService } from '@syncfusion/ej2-angular-maps'
import {
    Component,
    ViewEncapsulation,
    ViewChild,
    ElementRef,
} from '@angular/core';
import { world_map } from './world-map';
import { MapsComponent } from '@syncfusion/ej2-angular-maps';
@Component({
    imports: [
        MapsModule
    ],
    providers: [MarkerService, ZoomService],
    standalone: true,
    selector: 'app-container',
    template: ` <div class="control-section">
        <button id="button"
        (click)="getMinMaxValues()">GetMinMaxLatitudeLongitude</button>
        <p id="coordinatesDisplay" #coordinatesDisplay></p>
        <div align="center">
            <ejs-maps id="container" #maps style="display:block;"
            [zoomSettings]="zoom" [centerPosition]="centerPosition">
                <e-layers>
                    <e-layer [shapeData]='worldMap' [markerSettings]='markerSettings'></e-
layer>
                </e-layers>
            </ejs-maps>
        </div>` ,
    encapsulation: ViewEncapsulation.None,
})
export class AppComponent {
    @ViewChild('maps')
    public mapObj?: MapsComponent;
    @ViewChild('coordinatesDisplay') coordinatesDisplay!: ElementRef;
    public worldMap = world_map;
    public formatKey(key: string): string {
```

```

        if (key === 'minLatitude') {
            return 'Minimum Latitude';
        } else if (key === 'maxLatitude') {
            return 'Maximum Latitude';
        } else if (key === 'minLongitude') {
            return 'Minimum Longitude';
        } else if (key === 'maxLongitude') {
            return 'Maximum Longitude';
        }
        return key;
    }
    public getMinMaxValues() {
        let mapBoundCoordinates: any;
        mapBoundCoordinates = this.mapObj?.getMinMaxLatitudeLongitude();
        if (this.coordinatesDisplay &&
this.coordinatesDisplay.nativeElement) {
            const displayDiv = this.coordinatesDisplay.nativeElement;
            if (mapBoundCoordinates) {
                displayDiv.innerHTML = '';
                for (const key in mapBoundCoordinates) {
                    if (Object.hasOwnProperty.call(mapBoundCoordinates,
key)) {
                        const p = document.createElement('p');
                        const formattedKey = this.formatKey(key);
                        p.textContent = `${formattedKey}:
${mapBoundCoordinates[key]}`;
                        displayDiv.appendChild(p);
                    }
                }
            } else {
                displayDiv.textContent = 'No coordinates available';
            }
        }
    }
    public zoom: Object = {
        enable: true,
        zoomFactor: 7,
    };
    public centerPosition: Object = {
        latitude: 21.815447,
        longitude: 80.1932,
    };
    public markerSettings: Object = [
        {
            visible: true,
            height: 25,
            width: 25,
            shape: 'Circle',
            animationDuration: 1500,
            dataSource: [
                {
                    latitude: 22.572646,
                    longitude: 88.363895,
                },
                {
                    latitude: 25.0700428,
                    longitude: 67.2847875,
                },
            ],
        },
    ];

```

```

    },
    ],
  ];
}

```

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 MD034 -->

Ej1 api migration in Angular Maps component

This article describes the API migration process of Maps component from Essential JS 1 to Essential JS 2.

Size Customization

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Height | Not Applicable | **Property:** *height*
<ejs-maps id="maps" height="150px"></ejs-maps>|

| Width | Not Applicable | **Property:** *width*
<ejs-maps id="maps" width="150px"></ejs-maps>|

Title and Subtitle Customization

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Title Text | Not Applicable | **Property:** *title.text*
<ejs-maps id="maps" [titleSettings]="titleSettings"></ejs-maps>
 TS: public titleSettings: Object = { text: "Maps title" }|

| Subtitle Text | Not Applicable | **Property:** *title.subtitle.text*
<ejs-maps id="maps" [titleSettings]="titleSettings"></ejs-maps>
 TS: public titleSettings: Object = { subTitleSettings: { text: "Maps sub title" } }|

| Title Alignment | Not Applicable | **Property:** *title.alignment*
<ejs-maps id="maps" [titleSettings]="titleSettings"></ejs-maps>
 TS: public titleSettings: Object = { alignment: "Center" }|

| Subtitle Alignment | Not Applicable | **Property:** *title.subtitle.alignment*
id="maps" [titleSettings]="titleSettings"></ejs-maps>
 TS: public titleSettings: Object = { subTitleSettings: { alignment: "Center" } }|

<!-- markdownlint-disable MD038 -->

Zooming Customization

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Enable | **Property:** `zoomSettings.enableZoom`
`<ej-map id="container" [zoomSettings.enableZoom]="true"></ej-map>` | **Property:** `zoomSettings.enable`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { enable: true } |

| Minimum Zoom | **Property:** `zoomSettings.minValue`
`<ej-map id="container" [zoomSettings.minValue]=2></ej-map>` | **Property:** `zoomSettings.minZoom`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { minZoom: 2 } |

| Maximum Zoom | **Property:** `zoomSettings.maxValue`
`<ej-map id="container" [zoomSettings.maxValue]=5></ej-map>` | **Property:** `zoomSettings.maxZoom`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { maxZoom: 5 } |

| Mouse Wheel Zoom | **Property:** `zoomSettings.enableMouseWheelZoom`
`<ej-map id="container" [zoomSettings.enableMouseWheelZoom]="true"></ej-map>` | **Property:** `zoomSettings.mouseWheelZoom`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { mouseWheelZoom: true } |

| Double Click Zoom | Not Applicable | **Property:** `zoomSettings.doubleClickZoom`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { doubleClickZoom: true } |

| Pinch Zoom | Not Applicable | **Property:** `zoomSettings.pinchZooming`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { pinchZooming: true } |

| Single Click Zoom | **Property:** `zoomSettings.enableZoomOnSelection`
`<ej-map id="container" [zoomSettings.enableZoomOnSelection]="true"></ej-map>` | **Property:** `zoomSettings.zoomOnClick`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { zoomOnClick: true } |

| Zoom Factor | **Property:** `zoomSettings.factor`
`<ej-map id="container" [zoomSettings.factor]=2></ej-map>` | **Property:** `zoomSettings.zoomFactor`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { zoomFactor: 2 } |

| Toolbars | Not Applicable | **Property:** `zoomSettings.toolbars`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { toolbars: ["ZoomIn", "ZoomOut"] } |

| Toolbar Orientation | Not Applicable | **Property:** `zoomSettings.toolBarOrientation`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { toolBarOrientation: "Horizontal" } |

| Toolbar Vertical Alignment | Not Applicable | **Property:** `zoomSettings.verticalAlignment`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { verticalAlignment: "Center" } |

| Toolbar Horizontal Alignment | Not Applicable | **Property:** `zoomSettings.horizontalAlignment`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { horizontalAlignment: "Center" } |

| Toolbar Highlight Color | Not Applicable | **Property:** `zoomSettings.highlightColor`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { highlightColor: "#e61576" } |

| Toolbar Selection Color | Not Applicable | **Property:** `zoomSettings.selectionColor`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { selectionColor: "#e61576" } |

| Toolbar Fill Color | Not Applicable | **Property:** `zoomSettings.color`
`<ejs-maps id="maps" [zoomSettings]="zoomSettings"></ejs-maps>` **TS:** public zoomSettings: Object = { color: "#e61576" } |

Layer Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Type | Not Applicable | **Property:** `layers.type`
`<ejs-maps id="maps"> <e-layers><e-layer type="Layer"></e-layer></e-layers></ejs-maps>` |

| Layer Type | **Property:** `layers.layerType`
`<ej-map id="container"><e-layers><e-layer layerType="Geometry"></e-layer></e-layers></ej-map>` | **Property:** `layers.layerType`
`<ejs-maps id="maps"><e-layers><e-layer layerType="Geometry"></e-layer></e-layers></ejs-maps>` |

| Visible | Not Applicable | **Property:** `layers.visible`
`<ejs-maps id="maps"> <e-layers><e-layer visible="true"></e-layer></e-layers></ejs-maps>` |

| Bing Map Type | **Property:** `layers.bingMapType`
`<ej-map id="container"><e-layers><e-layer bingMapType="Aerial"></e-layer></e-layers></ej-map>` | **Property:** `layers.bingMapType`
`<ejs-maps id="maps"><e-layers><e-layer bingMapType="Aerial"></e-layer></e-layers></ejs-maps>` |

| Bing Map Key | **Property:** `layers.key`
`<ej-map id="container"><e-layers><e-layer key=""></e-layer></e-layers></ej-map>` | **Property:** `layers.key`
`<ejs-maps id="maps"><e-layers><e-layer key=""></e-layer></e-layers></ejs-maps>` |

| URL Template | **Property:** `layers.urlTemplate`
`<ej-map id="container"><e-layers><e-layer urlTemplate="http://a.tile.openstreetmap.org/level/tileX/tileY.png"></e-layer></e-layers></ej-map>` | **Property:** `layers.urlTemplate`
`<ejs-maps id="maps"><e-layers><e-layer urlTemplate="http://a.tile.openstreetmap.org/level/tileX/tileY.png"></e-layer></e-layers></ejs-maps>` |

| Shape Data | **Property:** `layers.shapeData`
`<ej-map id="container"><e-layers><e-layer [shapeData]="WorldMap"></e-layer></e-layers></ej-map>` | **Property:**

layers.shapeData
`<ej-maps id="maps"> <e-layers><e-layer
[shapeData]="WorldMap"></e-layer></e-layers></ej-maps>`

| Data Source | **Property:** *layers.dataSource*
`<ej-map id="container"><e-layers><e-layer
[dataSource]="PopulationData"></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"> <e-layers><e-layer
[dataSource]="PopulationData"></e-layer></e-layers></ej-maps>`

| Query | Not Applicable | **Property:** *layers.query*
`<ej-maps id="maps"> <e-layers><e-layer
query=""></e-layer></e-layers></ej-maps>`

| Shape Data Path | **Property:** *layers.shapeDataPath*
`<ej-map id="container"><e-layers><e-layer
shapeDataPath="Continent"></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"> <e-layers><e-layer
shapeDataPath="Continent"></e-layer></e-layers></ej-maps>`

| Shape Property Path | **Property:** *layers.shapePropertyPath*
`<ej-map id="container"><e-layers><e-layer
shapePropertyPath="Continent"></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"> <e-layers><e-layer
shapePropertyPath="Continent"></e-layer></e-layers></ej-maps>`

| Layer Animation | Not Applicable | **Property:** *layers.animationDuration*
`<ej-maps id="maps"> <e-layers><e-layer
animationDuration="100"></e-layer></e-layers></ej-maps>`

Shape Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Shape Fill | **Property:** *layers.shapeSettings.fill*
`<ej-map id="container"><e-layers
shapeSettings.fill="#9CBF4E"></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"><e-layers><e-layer [shapeSettings]
='shapeSettings'></e-layer> </e-layers></ej-maps>` **TS:** `public shapeSettings: Object = {
fill: "red" }`

| Shape Palette | **Property:** *layers.shapeSettings.colorPalette*
`<ej-map id="container"><e-layers><e-layer
shapeSettings.colorPalette="customPalette"></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"><e-layers><e-layer
[shapeSettings] ='shapeSettings'></e-layer> </e-layers></ej-maps>` **TS:** `public
shapeSettings: Object = { palette: ["red", "green"] }`

| Shape Point Radius | Not Applicable | **Property:** *layers.shapeSettings.circleRadius*
`<ej-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ej-maps>` **TS:** `public shapeSettings: Object = { circleRadius: 10 }`

| Shape Color Value Path | **Property:** *layers.shapeSettings.colorValuePath*
`<ej-map id="container"><e-layers shapeSettings.colorValuePath="Sales"><e-layer></e-layer></e-layers></ej-map>` | **Property:**
`<ej-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ej-maps>` **TS:** `public shapeSettings: Object = { colorValuePath: "country" }`

| Shape Value Path | **Property:** *layers.shapeSettings.valuePath*
`<ej-map id="container"><e-layers><e-layer shapeSettings.valuePath="Sales"></e-layer></e-layers></ej-map>` | **Property:** *layers.shapeSettings.valuePath*
`<ejs-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ejs-maps>` **TS:** `public shapeSettings: Object = { valuePath: "Sales" }`

| Shape DashArray | Not Applicable | **Property:** *layers.shapeSettings.dashArray*
`<ejs-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ejs-maps>` **TS:** `public shapeSettings: Object = { dashArray: "5,3" }`

| Shape Opacity | Not Applicable | **Property:** *layers.shapeSettings.opacity*
`<ejs-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ejs-maps>` **TS:** `public shapeSettings: Object = { opacity: 0.5 }`

| Range Color Mapping | **Property:** *layers.shapeSettings.colorMappings.rangeColorMapping*
`<ej-map id="container"><e-layers><e-layer [shapeSettings.colorMappings]="colorMapping"></e-layer></e-layers></ej-map>` **TS:** `public colorMapping: any = { rangeColorMapping: [{ from: '400', to: '600', color: '#C6C35C' }] }` | **Property:** *layers.shapeSettings.colorMapping*
`<ejs-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ejs-maps>` **TS:** `public shapeSettings: Object = { colorMapping: [{from: '400', to: '600', color: "green"}] }`

| Equal Color Mapping | **Property:** *layers.shapeSettings.colorMappings.equalColorMapping*
`<ej-map id="container"><e-layers><e-layer [shapeSettings.colorMappings]="colorMapping"></e-layer></e-layers></ej-map>` **TS:** `public colorMapping: any = { equalColorMapping: [{ value: "Ckinton", color: '#C6C35C' }] }` | **Property:** *layers.shapeSettings.colorMapping*
`<ejs-maps id="maps"><e-layers><e-layer [shapeSettings] ='shapeSettings'></e-layer> </e-layers></ejs-maps>` **TS:** `public shapeSettings: Object = { colorMapping: [{ value: "Ckinton", color: '#C6C35C' }] }`

Marker Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Marker Data Source | **Property:** *layers.markers*
`<ej-map id="container"><e-layers><e-layer [markers]="markers"></e-layer></e-layers></ej-map>` **TS:** `public markers: any = [{ latitude: 42, longitude: -93, label: "Iowa" }]` | **Property:** *layers.markerSettings.dataSource*
`<ejs-maps id="maps"><e-layers><e-layer [markerSettings] = 'markerSettings'></e-layers></ejs-maps>` **TS:** `public markerSettings: Object = [{ dataSource: [{ latitude: 49.95121990866204, longitude: 18.468749999999998 }] }`

| Marker Template | **Property:** *layers.markerTemplate*
`<div id="Template" style="background-image:url;margin-left:5px;height:25px;width:80px;margin-top:15px;"></div>` `<ej-map id="container"><e-layers><e-layer`

markerTemplate="template"></e-layer></e-layers></ej-map> | **Property:**
layers.markerSettings.template

 <ejs-maps id="maps"><e-layers><e-layer></e-layer>
 [markerSettings] = 'markerSettings'> </e-layers></ej-maps>
 TS:
 public
 markerSettings: Object = [{
 template: "template"
}]

| Marker Visible | Not Applicable | **Property:** *layers.markerSettings.visible*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 visible: true
}]

| Marker Fill | Not Applicable | **Property:** *layers.markerSettings.fill*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 fill: "red"
}]

| Marker Height | Not Applicable | **Property:** *layers.markerSettings.height*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 height: 10
}]

| Marker Width | Not Applicable | **Property:** *layers.markerSettings.width*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 width: 10
}]

| Marker Shape | Not Applicable | **Property:** *layers.markerSettings.shape*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 shape: "Circle"
}]

| Marker ImageURL | Not Applicable | **Property:** *layers.markerSettings.imageUrl*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 imageUrl:
 "http://js.syncfusion.com/demos/web/Images/map/pin.png"
}]

| Marker Opacity | Not Applicable | **Property:** *layers.markerSettings.opacity*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 opacity: 0.5
}]

| Marker Legend Text | Not Applicable | **Property:** *layers.markerSettings.legendText*

 <ejs-
 maps id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-
 layers></ej-maps>
 TS:
 public markerSettings: Object = [{
 legendText:
 "China"
}]

| Marker Offset | Not Applicable | **Property:** *layers.markerSettings.offset*

 <ejs-maps
 id="maps"><e-layers><e-layer></e-layer> [markerSettings] = 'markerSettings'> </e-layers></ej-
 maps>
 TS:
 public markerSettings: Object = [{
 offset: 10
}]

| Marker Animation Duration | Not Applicable | **Property:**
layers.markerSettings.animationDuration

 <ejs-maps id="maps"><e-layers><e-layer></e-
 layer> [markerSettings] = 'markerSettings'> </e-layers></ej-maps>
 TS:
 public
 markerSettings: Object = [{
 animationDuration: 2000
}]

| Marker Animation Delay | Not Applicable | **Property:**
layers.markerSettings.animationDelay

 <ejs-maps id="maps"><e-layers><e-layer></e-

layer [markerSettings] = 'markerSettings' </e-layers></ejs-maps>
 TS:
 public markerSettings: Object = [{
 animationDelay: 100
}]

| Marker DashArray | Not Applicable | **Property:** layers.markerSettings.dashArray

 <ejs-maps id="maps"><e-layers><e-layer></e-layer [markerSettings] = 'markerSettings'> </e-layers></ejs-maps>
 TS:
 public markerSettings: Object = [{
 dashArray: "2,3"
}]

| Marker Selection | Not Applicable | **Property:** layers.markerSettings.selectionSettings

 <ejs-maps id="maps"><e-layers><e-layer></e-layer [markerSettings] = 'markerSettings'> </e-layers></ejs-maps>
 TS:
 public markerSettings: Object = [{
 selectionSettings: {enable: true, fill: "lime", opacy:0.5, enaleMultiSelect: false }
}]

| Marker Highlight | Not Applicable | **Property:** layers.markerSettings.highlightSettings

 <ejs-maps id="maps"><e-layers><e-layer></e-layer [markerSettings] = 'markerSettings'> </e-layers></ejs-maps>
 TS:
 public markerSettings: Object = [{
 highlightSettings: {enable: true, fill: "lime", opacy:0.5 }
}]

| Marker Tooltip | Not Applicable | **Property:** layers.markerSettings.tooltipSettings

 <ejs-maps id="maps"><e-layers><e-layer></e-layer [markerSettings] = 'markerSettings'> </e-layers></ejs-maps>
 TS:
 public markerSettings: Object = [{
 tooltipSettings: {enable: true, fill: "lime", valuePath: "State", template: "tooltipTemplate"}
}]

Bubble Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Visible | **Property:** layers.bubbleSettings.visible

 <ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>
 TS:
 public bubbleSettings: any = { showBubble = true }; | **Property:** layers.bubbleSettings.visible

 <ejs-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ejs-maps>
 TS:
 public bubbleSettings: Object = [{ visible : true }];

| ValuePath | **Property:** layers.bubbleSettings.valuePath

 <ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>
 TS:
 public bubbleSettings: any = { valuePath = "Sales" }; | **Property:** layers.bubbleSettings.valuePath

 <ejs-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ejs-maps>
 TS:
 public bubbleSettings: Object = [{ valuePath : "Sales" }];

| MinValue | **Property:** layers.bubbleSettings.minValue

 <ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>
 TS:
 public bubbleSettings: any = { minValue = 10 }; | **Property:** layers.bubbleSettings.minRadius

 <ejs-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ejs-maps>
 TS:
 public bubbleSettings: Object = [{ minRadius : 10 }];

| MaxValue | **Property:** layers.bubbleSettings.maxValue

 <ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>
 TS:
 public bubbleSettings: any = { maxValue = 20 }; | **Property:**

layers.bubbleSettings.maxRadius
`<ej-s-maps id="maps"> <e-layers><e-layer
 bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-maps> </> TS:
 public
 bubbleSettings: Object = [{ maxRadius : 20 }];|`

| Bubble Type | Not Applicable | **Property:** *layers.bubbleSettings.bubbleType*
`<ej-s-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-
 maps> </> TS:
 public bubbleSettings: Object = [{ bubbleType : "Circle" }];|`

| Color | **Property:** *layers.bubbleSettings.color*
`<ej-map id="container"><e-layers><e-
 layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map> </> TS:
 public
 bubbleSettings: any = { color = "green" }; | Property: layers.bubbleSettings.fill
<ej-s-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-
 maps> </> TS:
 public bubbleSettings: Object = [{ fill : "red" }];|`

| Opacity | **Property:** *layers.bubbleSettings.bubbleOpacity*
`<ej-map id="container"><e-
 layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map> </> TS:

 public bubbleSettings: any = { bubbleOpacity = 0.4 }; | Property:
layers.bubbleSettings.opacity
<ej-s-maps id="maps"> <e-layers><e-layer
 bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-maps> </> TS:
 public
 bubbleSettings: Object = [{ bubbleOpacity : 0.5 }];|`

| Color Value Path | **Property:** *layers.bubbleSettings.colorValuePath*
`<ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-
 layers></ej-map> </> TS:
 public bubbleSettings: any = { colorValuePath = "Sales" }; | Property:
layers.bubbleSettings.colorValuePath
<ej-s-maps id="maps"> <e-layers><e-layer
 bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-maps> </> TS:
 public
 bubbleSettings: Object = [{ colorValuePath : "Sales" }];|`

| Enable Tooltip | **Property:** *layers.bubbleSettings.showTooltip*
`<ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-
 layers></ej-map> </> TS:
 public bubbleSettings: any = { showTooltip = true }; | Property:
layers.bubbleSettings.tooltipSettings.visible
<ej-s-maps id="maps"> <e-layers><e-layer
 bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-maps> </> TS:
 public
 bubbleSettings: Object = [{
 tooltipSettings: { visible : true }
 }];|`

| Tooltip Template | **Property:** *layers.bubbleSettings.tooltipTemplate*
`<ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-
 layers></ej-map>
<div id="template">
..
</div>
 TS:
 public
 bubbleSettings: any = { tooltipTemplate = "template" }; | Property:
layers.bubbleSettings.tooltipSettings.template
<div id="template">

..
</div>
<ej-s-maps id="maps"> <e-layers><e-layer
 bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-s-maps>
 TS:
 public
 bubbleSettings: Object = [{
 tooltipSettings: { template: "template" }
 }];|`

| Bubble Selection | Not Applicable | **Property:** *layers.bubbleSettings.selectionSettings*
`<ej-s-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-
 layers></ej-s-maps> </> TS:
 public bubbleSettings: Object = [{
 selectionSettings: { fill:
 "red", opacity: 1, enable: true, enableMultiSelect: false }
 }];|`

| Bubble Highlight | Not Applicable | **Property:** *layers.bubbleSettings.highlightSettings*
`<ej-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-maps>` **TS:** `public bubbleSettings: Object = [{ highlightSettings: { fill: "red", opacity: 1, enable: true } }];` |

| Range Color Mapping | **Property:** *layers.bubbleSettings.colorMappings.rangeColorMapping*
`<ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>` **TS:** `public bubbleSettings: any = { colorMappings: { rangeColorMapping: [{ from: 10, to: 40, color: "red" }] } }`; | **Property:** *layers.bubbleSettings.colorMapping*
`<ej-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-maps>` **TS:** `public bubbleSettings: Object = [{ colorMapping: { from: 60000, to: 10000, color: "red" } }];` |

| Equal Color Mapping | **Property:** *layers.bubbleSettings.colorMappings.equalColorMapping*
`<ej-map id="container"><e-layers><e-layer [bubbleSettings]="bubbleSettings"></e-layer></e-layers></ej-map>` **TS:** `public bubbleSettings: any = { colorMappings: { equalColorMapping: [{ value: "china", color: "red" }] } }`; | **Property:** *layers.bubbleSettings.colorMapping*
`<ej-maps id="maps"> <e-layers><e-layer bubbleSettings="bubbleSettings"></e-layer></e-layers></ej-maps>` **TS:** `public bubbleSettings: Object = [{ colorMapping: { value: "China", color: "red" } }];` |

DataLabel Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Visible | **Property:** *layers.labelSettings.showLabels*
`<ej-map id="container"><e-layers><e-layer [labelSettings.showLabels]=true></e-layer></e-layers></ej-map>` | **Property:** *layers.dataLabelSettings.visible*
`<ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>` **TS:** `public datalabelsettings: Object = { visible: true };` |

| Label Path | **Property:** *layers.labelSettings.labelPath*
`<ej-map id="container"><e-layers><e-layer labelSettings.labelPath="Continent"></e-layer></e-layers></ej-map>` | **Property:** *layers.dataLabelSettings.labelPath*
`<ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>` **TS:** `public datalabelsettings: Object = { labelPath: "Continent" };` |

| Enable Smart Label | **Property:** *layers.labelSettings.enableSmartLabel*
`<ej-map id="container"><e-layers><e-layer [labelSettings.enableSmartLabels]=true></e-layer></e-layers></ej-map>` | Not Applicable |

| Smart Label Size | **Property:** *layers.labelSettings.smartLabelSize*
`<ej-map id="container"><e-layers><e-layer [labelSettings.smartLabelSize]=20></e-layer></e-layers></ej-map>` | Not Applicable |

| Label Length | **Property:** *layers.labelSettings.labelLength*
`<ej-map id="container"><e-layers><e-layer [labelSettings.labelLength]=20></e-layer></e-layers></ej-map>` | Not Applicable |

| Opacity | Not Applicable | **Property:** *layers.dataLabelSettings.opacity*
 <ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>
 TS:
 public datalabelsettings: Object = {
 opacity: 0.5
 };|

| Smart Label Mode | Not Applicable | **Property:** *layers.dataLabelSettings.smartLabelMode*
 <ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>
 TS:
 public datalabelsettings: Object = {
 smartLabelsMode: "Trim"
 };|

| InterSectAction | Not Applicable | **Property:** *layers.dataLabelSettings.intersectionAction*
 <ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>
 TS:
 public datalabelsettings: Object = {
 intersectionAction: "None"
 };|

| Template | Not Applicable | **Property:** *layers.dataLabelSettings.template*
 <ej-maps id="maps"> <e-layers><e-layer [datalabelsettings]="datalabelsettings"></e-layer></e-layers></ej-maps>
 <div id="template">
 //..
</div>
 TS:
 public datalabelsettings: Object = {
 template: "template"
 };|

Legend Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Visible | **Property:** *layers.legendSettings.showLegend*
 <ej-map id="container"><e-layers><e-layer [legendSettings.showLegend]=true></e-layer></e-layers></ej-map> | **Property:** *legendSettings.visible*
 <ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>
 TS:
 public legendSettings: Object = {
 visible: true
 };|

| Toggle Visibility | **Property:** *layers.legendSettings.toggleVisibility*
 <ej-map id="container"><e-layers><e-layer [legendSettings.toggleVisibility]=true></e-layer></e-layers></ej-map> | **Property:** *legendSettings.toggleVisibility*
 <ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>
 TS:
 public legendSettings: Object = {
 toggleVisibility: true
 };|

| Legend Location X | **Property:** *layers.legendSettings.positionX*
 <ej-map id="container"><e-layers><e-layer [legendSettings.positionX]=250></e-layer></e-layers></ej-map> | **Property:** *legendSettings.location*
 <ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>
 TS:
 public legendSettings: Object = {
 location: { x: 10 }
 };|

| Legend Location Y | **Property:** *layers.legendSettings.positionY*
 <ej-map id="container"><e-layers><e-layer [legendSettings.positionY]=250></e-layer></e-layers></ej-map> | **Property:** *legendSettings.location*
 <ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>
 TS:
 public legendSettings: Object = {
 location: { y: 20 }
 };|

| Legend Type | **Property:** *layers.legendSettings.type*
 <ej-map id="container"><e-layers><e-layer legendSettings.type="Layers"></e-layer></e-layers></ej-map> | **Property:**

`legendSettings.type

<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 type: "Layers"
 }; |`

| Label Orientation | **Property:** `layers.legendSettings.labelOrientation

` `<ej-map id="container"><e-layers><e-layer legendSettings.labelOrientation="Vertical"></e-layer></e-layers></ej-map>` | Not Applicable |

| Legend Title | **Property:** `layers.legendSettings.title

` `<ej-map id="container"><e-layers><e-layer legendSettings.title="Unio territories of India"></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.title

` `<ej-maps id="maps"`

`[legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 title: { text: "Union territories of India" }
 }; |`

| Legend Mode | **Property:** `layers.legendSettings.mode

` `<ej-map id="container"><e-layers><e-layer legendSettings.mode="Default"></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.mode

` `<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 mode: "Default"
 }; |`

| Legend Position | **Property:** `layers.legendSettings.position

` `<ej-map id="container"><e-layers><e-layer legendSettings.position="topleft"></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.position

` `<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 position: "Top"
 }; |`

| Legend DockOnMap | **Property:** `layers.legendSettings.dockOnMap

` `<ej-map id="container"><e-layers><e-layer [legendSettings.dockOnMap]=true></e-layer></e-layers></ej-map>` | Not Applicable |

| Legend Alignment | **Property:** `layers.legendSettings.dockPosition

` `<ej-map id="container"><e-layers><e-layer legendSettings.dockPosition="right"></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.alignment

` `<ej-maps id="maps"` `[legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 alignment: "Center"
 }; |`

| Legend Left Label | **Property:** `layers.legendSettings.leftLabel

` `<ej-map id="container"><e-layers><e-layer legendSettings.leftLael="1000M"></e-layer></e-layers></ej-map>` | Not Applicable |

| Legend Right Label | **Property:** `layers.legendSettings.rightLabel

` `<ej-map id="container"><e-layers><e-layer legendSettings.rightLabel="300M"></e-layer></e-layers></ej-map>` | Not Applicable |

| Legend Shape | **Property:** `layers.legendSettings.icon

` `<ej-map id="container"><e-layers><e-layer legendSettings.icon="rectangle"></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.shape

` `<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `
 public legendSettings: Object = {
 shape: "Circle"
 }; |`

| Legend Shape Height | **Property:** `layers.legendSettings.iconHeight

` `<ej-map id="container"><e-layers><e-layer [legendSettings.iconHeight]=10></e-layer></e-layers></ej-map>` | **Property:** `legendSettings.shapeHeight

` `<ej-maps id="maps"`

[legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 shapeHeight: 10
 }; |

| Legend Shape Width | **Property:** *layers.legendSettings.iconWidth*

 <ej-map id="container"><e-layers><e-layer [legendSettings.iconWidth]=10></e-layer></e-layers></ej-map> | **Property:** *legendSettings.shapeWidth*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 shapeWidth: 10
 }; |

| Height | **Property:** *layers.legendSettings.height*

 <ej-map id="container"><e-layers><e-layer [legendSettings.height]=10></e-layer></e-layers></ej-map> | **Property:** *legendSettings.width*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 height:20
 }; |

| Width | **Property:** *layers.legendSettings.width*

 <ej-map id="container"><e-layers><e-layer [legendSettings.width]=10></e-layer></e-layers></ej-map> | **Property:** *legendSettings.width*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 width: 20
 }; |

| Show Labels | **Property:** *layers.legendSettings.showLabels*

 <ej-map id="container"><e-layers><e-layer [legendSettings.showLabels]=true></e-layer></e-layers></ej-map> | Not Applicable |

| Background | Not Applicable | **Property:** *legendSettings.background*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 background: "transparent"
 }; |

| Label Position | Not Applicable | **Property:** *legendSettings.labelPosition*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 labelPosition: "After"
 }; |

| Label Display Mode | Not Applicable | **Property:** *legendSettings.labelDisplayMode*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 labelDisplayMode: "Trim"
 }; |

| Legend Orientation | Not Applicable | **Property:** *legendSettings.orientation*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 orientation: "Horizontal"
 }; |

| Legend Item Fill | Not Applicable | **Property:** *legendSettings.fill*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 fill: "red"
 }; |

| Legend Shape Padding | Not Applicable | **Property:** *legendSettings.shapePadding*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 shapePadding: 20
 }; |

| Legend Shape Border Color | Not Applicable | **Property:** *legendSettings.shapeBorder.color*

 <ejs-maps id="maps" [legendSettings]="legendSettings"></ejs-maps>
 TS:
 public legendSettings: Object = {
 shapeBorder: { color: "red" }
 }; |

| Legend Shape Border Width | Not Applicable | **Property:** *legendSettings.shapeBorder.width*
`<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `public legendSettings: Object = { shapeBorder: { width: 2 } }`;

| Inverter Pointer | Not Applicable | **Property:** *legendSettings.invertedPointer*
`<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `public legendSettings: Object = { invertedPointer: true }`;

| Item Text Style | Not Applicable | **Property:** *legendSettings.textStyle*
`<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `public legendSettings: Object = { textStyle: { fontWeight: "400", size: "14px" } }`;

| Title Style | Not Applicable | **Property:** *legendSettings.textStyle*
`<ej-maps id="maps" [legendSettings]="legendSettings"></ej-maps>` **TS:** `public legendSettings: Object = { titleStyle: { fontWeight: "400", size: "14px" } }`;

Highlight And Selection Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Highlight Fill | **Property:** *layers.shapeSettings.highlightColor*
`<ej-map id="container"><e-layers><e-layer shapeSettings.highlightColor="#BC5353"></e-layer></e-layers></ej-map>` | **Property:** *fill*
`<ej-maps id="maps"><e-layers><e-layer [highlightSettings]='highlightSettings'></e-layer></e-layers></ej-maps>` **TS:** `public highlightSettings : Object = { fill: "green" };`

| Enable Highlight | **Property:** *layers.enableMouseHover*
`<ej-map id="container"><e-layers><e-layer [enableMouseHover]=true></e-layer></e-layers></ej-map>` | **Property:** *enable*
`<ej-maps id="maps"><e-layers><e-layer [highlightSettings]='highlightSettings'></e-layer></e-layers></ej-maps>` **TS:** `public highlightSettings : Object = { enable: true };`

| Highlight Border Color | **Property:** *layers.shapeSettings.highlightStroke*
`<ej-map id="container"><e-layers><e-layer shapeSettings.highlightStroke="red"></e-layer></e-layers></ej-map>` | **Property:** *layers.highlightSettings.border.color*
`<ej-maps id="maps"><e-layers><e-layer [highlightSettings]='highlightSettings'></e-layer></e-layers></ej-maps>` **TS:** `public highlightSettings : Object = { border { color: "green" } };`

| Highlight Border Width | **Property:** *layers.shapeSettings.highlightBorderWidth*
`<ej-map id="container"><e-layers><e-layer [shapeSettings.highlightBorderWidth]=2></e-layer></e-layers></ej-map>` | **Property:** *layers.highlightSettings.border.width*
`<ej-maps id="maps"><e-layers><e-layer [highlightSettings]='highlightSettings'></e-layer></e-layers></ej-maps>` **TS:** `public highlightSettings : Object = { border { width: 2 } };`

| Highlight Opacity | Not Applicable | **Property:** *layers.layers.highlightSettings.opacity*
`<ej-maps id="maps"><e-layers><e-layer [highlightSettings]='highlightSettings'></e-layer></e-layers></ej-maps>` **TS:** `public highlightSettings : Object = { opacity: 0.5 };`

| Selection Fill | **Property:** *layers.shapeSettings.selectionColor*
`<ej-map id="container"><e-layers><e-layer shapeSettings.selectionColor="blue"></e-layer></e-layers></ej-map>` |
Property: *layers.selectionSettings.fill*
`<ejs-maps id="maps" ><e-layers><e-layer [selectionSettings] ='selectionSettings'></e-layer></e-layers></ejs-maps>` **TS:** `public selectionSettings : Object = { fill: "green" };|`

| Selection Enable | **Property:** *layers.enableSelection*
`<ej-map id="container"><e-layers><e-layer [enableSelection]=true></e-layer></e-layers></ej-map>` | **Property:** *layers.selectionSettings.enable*
`<ejs-maps id="maps" ><e-layers><e-layer [selectionSettings] ='selectionSettings'></e-layer></e-layers></ejs-maps>` **TS:** `public selectionSettings : Object = { enable: true };|`

| Selection Border Width | **Property:** *layers.selectionSettings.selectionStrokeWidth*
`<ej-map id="container"><e-layers><e-layer selectionSettings.selectionStrokeWidth="2"></e-layer></e-layers></ej-map>` | **Property:** *layers.selectionSettings.border.width*
`<ejs-maps id="maps"><e-layers><e-layer [selectionSettings] ='selectionSettings'></e-layer></e-layers></ejs-maps>` **TS:** `public selectionSettings : Object = { border: { width: 2 } };|`

| Selection Border Color | **Property:** *layers.selectionSettings.selectionStroke*
`<ej-map id="container"><e-layers><e-layer selectionSettings.selectionStroke="red"></e-layer></e-layers></ej-map>` | **Property:** *layers.selectionSettings.border.color*
`<ejs-maps id="maps"><e-layers><e-layer [selectionSettings] ='selectionSettings'></e-layer></e-layers></ejs-maps>` **TS:** `public selectionSettings : Object = { border: { color: "green" } };|`

| Selection Opacity | Not Applicable | **Property:** *layers.selectionSettings.opacity*
`<ejs-maps id="maps"><e-layers><e-layer [selectionSettings] ='selectionSettings'></e-layer></e-layers></ejs-maps>` **TS:** `public selectionSettings : Object = { opacity: 0.7 };|`

Navigation Line Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Visible | Not Applicable | **Property:** *layers.navigationLineSettings.visible*
`<ejs-maps id="maps"><e-layers><e-layer <e-layersettings-navigationlines> <e-layersettings-navigationline visible="true" ></e-layersettings-navigationline> </e-layersettings-navigationlines> </e-layer></ejs-maps>` |

| Width | Not Applicable | **Property:** *layers.navigationLineSettings.width*
`<ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>` **TS:** `public navigationLineSettings: Object[] = [{
 width: 2
 }]; |`

| Longitude | Not Applicable | **Property:** *layers.navigationLineSettings.longitude*
`<ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>` **TS:** `public navigationLineSettings: Object[] = [{
 latitude: [37.6276571, -14.2350]
 }]; |`

| Latitude | Not Applicable | **Property:** *layers.navigationLineSettings.latitude*
`<ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-`

`layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{

longitude: [-74.0060, -51.9253]
 }]; |`

| DashArray | Not Applicable | **Property:** `layers.navigationLineSettings.dashArray

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
dashArray: "5,3"
 }]; |`

| Color | Not Applicable | **Property:** `layers.navigationLineSettings.color

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
color: "green"
 }]; |`

| Angle | Not Applicable | **Property:** `layers.navigationLineSettings.angle

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
angle: 180
 }]; |`

| Arrow Position | Not Applicable | **Property:** `layers.navigationLineSettings.arrow.position

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
arrow: { position: "Start" }
 }]; |`

| Show Arrow | Not Applicable | **Property:** `layers.navigationLineSettings.arrow.showArrow

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
arrow: { showArrow: true }
 }]; |`

| Arrow size | Not Applicable | **Property:** `layers.navigationLineSettings.arrow.size

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
arrow: { size: 2 }
 }]; |`

| Arrow Color | Not Applicable | **Property:** `layers.navigationLineSettings.arrow.color

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
arrow: { color: "red" }
 }]; |`

| Arrow Offset | Not Applicable | **Property:** `layers.navigationLineSettings.arrow.offSet

 <ejs-maps id="maps"><e-layers><e-layer [navigationLineSettings] = 'navigationLineSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public navigationLineSettings: Object[] = [{
arrow: { offset: 10 }
 }]; |`

Tooltip Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Tooltip Enable | **Property:** `layers.showTooltip

 <ej-map id="container"><e-layers><e-layer [showTooltip]=true></e-layer></e-layers></ej-map> | Property: layers.tooltipSettings.visible

 <ejs-maps id="maps"><e-layers><e-layer`

[tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
visible: true
 }; |

| Tooltip Template | **Property:** *layers.tooltipTemplate*

 <div id=template>
 \\.
</div>
<ej-map id="container"><e-layers><e-layer tooltipTemplate="template"></e-layer></e-layers></ej-map> | **Property:** *layers.tooltipSettings.template*

 <div id=template>
 \\.
</div>
<ejs-maps id="maps"><e-layers><e-layer [tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
template: "template"
 }; |

| Value Path | Not Applicable | **Property:** *layers.tooltipSettings.valuePath*

 <ejs-maps id="maps"><e-layers><e-layer [tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
valuePath: "Sales"
 }; |

| Format | Not Applicable | **Property:** *layers.tooltipSettings.format*

 <ejs-maps id="maps"><e-layers><e-layer [tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
format: "{\$State} {\$Population}"
 }; |

| Border Color | Not Applicable | **Property:** *layers.tooltipSettings.border.color*

 <ejs-maps id="maps"><e-layers><e-layer [tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
border: {color: "red" }
 }; |

| Border Width | Not Applicable | **Property:** *layers.tooltipSettings.border.width*

 <ejs-maps id="maps"><e-layers><e-layer [tooltipSettings]='tooltipSettings'></e-layers></e-layer></ejs-maps>
 TS:
 public tooltipSettings: Object = {
border: { width: 2 }
 }; |

Annotation Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Content | Not Applicable | **Property:** *legendSettings.annotations.content*

 <ejs-maps id="maps" [annotations]='annotations'></ejs-maps>
 TS:
 public annotations: Object[] = [{
content: ""
}]; |

| Location X | Not Applicable | **Property:** *legendSettings.annotations.x*

 <ejs-maps id="maps" [annotations]='annotations'></ejs-maps>
 TS:
 public annotations: Object[] = [{
x: "100px"
}]; |

| Location Y | Not Applicable | **Property:** *legendSettings.annotations.y*

 <ejs-maps id="maps" [annotations]='annotations'></ejs-maps>
 TS:
 public annotations: Object[] = [{
y: "250px"
}]; |

| Vertical Alignment | Not Applicable | **Property:** *legendSettings.annotations.verticalAlignment*

 <ejs-maps id="maps" [annotations]='annotations'></ejs-maps>
 TS:
 public annotations: Object[] = [{
verticalAlignment: "Center"
}]; |

| Horizontal Alignment | Not Applicable | **Property:** *legendSettings.annotations.horizontalAlignment*

 <ejs-maps id="maps" [annotations]='annotations'></ejs-maps>
 TS:
 public annotations: Object[] = [{
horizontalAlignment: "Center"
}]; |

| Zindex | Not Applicable | **Property:** *legendSettings.annotations.zIndex*
`<ej-maps id="maps" [annotations]='annotations'></ej-maps>` **TS:** `public annotations: Object[] = [{ zIndex: "-1"}];` |

Maps Other Properties Customization

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Projection Type | Not Applicable | **Property:** *projectionType*
`<ej-maps id="container" projectionType="Mercator"></ej-maps>` |

| Background | **Property:** *background*
`<ej-map id="container" background="red"></ej-map>` | **Property:** *background*
`<ej-maps id="container" background="red"></ej-maps>` |

| Enable Group Separator | **Property:** *enableGroupSeparator*
`<ej-map id="container" [enableGroupSeparator]=true></ej-map>` | **Property:** *useGroupingSeparator*
`<ej-maps id="container" [useGroupingSeparator]="separator"></ej-maps>` **TS:** `public separator: boolean: true;` |

| Base Layer Index | **Property:** *baseMapIndex*
`<ej-map id="container" [baseMapIndex]=0></ej-map>` | **Property:** *baseLayerIndex*
`<ej-maps id="container" [baseLayerIndex]="index"></ej-maps>` **TS:** `public index: number:0;` |

| locale | **Property:** *locale*
`<ej-map id="container" locale=""></ej-map>` | Not Applicable |

| Responsive | **Property:** *isResponsive*
`<ej-map id="container" [isResponsive]=true></ej-map>` | Not Applicable |

| Enable Pan | **Property:** *enablePan*
`<ej-map id="container" [enablePan]=true></ej-map>` | Not Applicable |

| Enable Navigation | **Property:** *navigationControl.enableNavigation*
`<ej-map id="container" [enableNavigation]=true></ej-map>` | Not Applicable |

| Navigation Orientation | **Property:** *navigationControl.orientation*
`<ej-map id="container" orientation=""></ej-map>` | Not Applicable |

| Navigation Dock Position | **Property:** *navigationControl.dockPosition*
`<ej-map id="container" dockPosition=""></ej-map>` | Not Applicable |

| Navigation Absolute Position | **Property:** *navigationControl.absolutePosition*
`<ej-map id="container" absolutePosition=""></ej-map>` | Not Applicable |

| Dragging Selection | **Property:** *draggingOnSelection*
`<ej-map id="container" [draggingOnSelection]=true></ej-map>` | Not Applicable |

| Resize | **Property:** *enableResize*
`<ej-map id="container" [enableResize]=true></ej-map>` | Not Applicable |

| Enable Animation | **Property:** *enableAnimation*
`<ej-map id="container" [enableAnimation]=true></ej-map>` | Not Applicable |

| Enable Layer Animation | **Property:** *enableLayerChangeAnimation*

 <ej-map id="container" [enableLayerAnimation]=true></ej-map> | Not Applicable |

| Center Position | **Property:** *centerPosition*

 <ej-map id="container" centerPosition=""></ej-map> | **Property:** *centerPosition*

 <ejs-maps id="container" centerPosition=""> </ejs-maps> |

Events

| **Behavior** | **API in Essential JS 1** | **API in Essential JS 2** |

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

| Shape Selected | **Property:** *shapeSelected*

 <ej-map id="container" (shapeSelected)="shapeSelected(\$event)"></ej-map>
 TS:
public shapeSelected(args){
} | **Property:** *shapeSelected*

 <ejs-maps id="maps" (shapeSelected)="shapeSelected(\$event)"></ejs-maps>
 TS:
public shapeSelected(args){
} |

| Marker Selected | **Property:** *markerSelected*

 <ej-map id="container" (markerSelected)="markerSelected(\$event)"></ej-map>
 TS:
public markerSelected(args){
} | **Property:** *markerClick*

 <ejs-maps id="maps" (markerClick)="markerClick(\$event)"></ejs-maps>
 TS:
public markerClick(args){
} |

| Marker Move | **Property:** *markerEnter*

 <ej-map id="container" (markerEnter)="markerEnter(\$event)"></ej-map>
 TS:
public markerEnter(args){
} | **Property:** *markerMouseMove*

 <ejs-maps id="maps" (markerMouseMove)="markerMouseMove(\$event)"></ejs-maps>
 TS:
public markerMouseMove(args){
} |

| Marker Leave | **Property:** *markerLeave*

 <ej-map id="container" (markerLeave)="markerLeave(\$event)"></ej-map>
 TS:
public markerLeave(args){
} | Not Applicable |

| Legend Item Rendering | **Property:** *legendItemRendering*

 <ej-map id="container" (legendItemRendering)="legendItemRendering(\$event)"></ej-map>
 TS:
public legendItemRendering(args){
} | Not Applicable |

| Display Text Rendering | **Property:** *displayTextRendering*

 <ej-map id="container" (displayTextRendering)="displayTextRendering(\$event)"></ej-map>
 TS:
public displayTextRendering(args){
} | **Property:** *dataLabelRendering*

 <ejs-maps id="maps" (dataLabelRendering)="dataLabelRendering(\$event)"></ejs-maps>
 TS:
public dataLabelRendering(args){
} |

| Legend Item Click | **Property:** *legendItemClick*

 <ej-map id="container" (legendItemClick)="legendItemClick(\$event)"></ej-map>
 TS:
public legendItemClick(args){
} | Not Applicable |

| Bubble Rendering | **Property:** *bubbleRendering*

 <ej-map id="container" (bubbleRendering)="bubbleRendering(\$event)"></ej-map>
 TS:
public bubbleRendering(args){
} | **Property:** *bubbleRendering*

 <ejs-maps id="maps" (bubbleRendering)="bubbleRendering(\$event)"></ejs-maps>
 TS:
public bubbleRendering(args){
} |

(bubbleRendering)="bubbleRendering(\$event)"></ej-maps>
 TS:
public
bubbleRendering(args){
 } |

| Shape Rendering | **Property:** *shapeRendering*

 <ej-map id="container"
(shapeRendering)="shapeRendering(\$event)"></ej-map>
 TS:
public
shapeRendering(args){
 } | **Property:** *shapeRendering*

 <ej-maps id="maps"
(shapeRendering)="shapeRendering(\$event)"></ej-maps>
 TS:
public
shapeRendering(args){
 } |

| Zoomed In | **Property:** *zoomedIn*

 <ej-map id="container"
(zoomedIn)="zoomedIn(\$event)"></ej-map>
 TS:
public zoomedIn(args){
 } | Not
Applicable |

| Render Completed | **Property:** *onRenderComplete*

 <ej-map id="container"
(onRenderComplete)="onRenderComplete(\$event)"></ej-map>
 TS:
public
onRenderComplete(args){
 } | **Property:** *loaded*

 <ej-maps id="maps"
(loaded)="loaded(\$event)"></ej-maps>
 TS:
public loaded(args){
 } |

| Panned | **Property:** *panned*

 <ej-map id="container" (panned)="panned(\$event)"></ej-
map>
 TS:
public panned(args){
 } | Not Applicable |

| zoomed Out | **Property:** *zoomedOut*

 <ej-map id="container"
(zoomedOut)="zoomedOut(\$event)"></ej-map>
 TS:
public zoomedOut(args){
 } |
Not Applicable |

| Mouse Over | **Property:** *mouseover*

 <ej-map id="container"
(mouseover)="mouseover(\$event)"></ej-map>
 TS:
public mouseover(args){
 } | Not
Applicable |

| Mouse Leave | **Property:** *mouseleave*

 <ej-map id="container"
(mouseleave)="mouseleave(\$event)"></ej-map>
 TS:
public mouseleave(args){
 } |
Not Applicable |

| Click | **Property:** *click*

 <ej-map id="container" (click)="click(\$event)"></ej-map>

TS:
public click(args){
 } | **Property:** *click*

 <ej-maps id="maps"
(click)="click(\$event)"></ej-maps>
 TS:
public click(args){
 } |

| Double Click | **Property:** *doubleClick*

 <ej-map id="container"
(doubleClick)="doubleClick(\$event)"></ej-map>
 TS:
public doubleClick(args){
 } |
Property: *doubleClick*

 <ej-maps id="maps" (doubleClick)="doubleClick(\$event)"></ej-
maps>
 TS:
public doubleClick(args){
 } |

| Right Click | **Property:** *rightClick*

 <ej-map id="container"
(rightClick)="rightClick(\$event)"></ej-map>
 TS:
public rightClick(args){
 } | **Property:**
rightClick

 <ej-maps id="maps" (rightClick)="rightClick(\$event)"></ej-maps>
 TS:

public rightClick(args){
 } |

| Initial Load | **Property:** *onLoad*

 <ej-map id="container"
(onLoad)="onLoad(\$event)"></ej-map>
 TS:
public onLoad(args){
 } | **Property:**
load

 <ej-maps id="maps" (load)="load(\$event)"></ej-maps>
 TS:
public
load(args){
 } |

| Before Print | Not Applicable | **Property:** *beforePrint*

 <ejs-maps id="maps"
(beforePrint)="beforePrint(\$event)"></ejs-maps>
 TS:
public beforePrint(args){
} |

| Resize | Not Applicable | **Property:** *resize*

 <ejs-maps id="maps"
(resize)="resize(\$event)"></ejs-maps>
 TS:
public resize(args){
} |

| Tooltip Render | Not Applicable | **Property:** *tooltipRender*

 <ejs-maps id="maps"
(tooltipRender)="tooltipRender(\$event)"></ejs-maps>
 TS:
public tooltipRender(args){

} |

| Item Selection | Not Applicable | **Property:** *itemSelection*

 <ejs-maps id="maps"
(itemSelection)="itemSelection(\$event)"></ejs-maps>
 TS:
public itemSelection(args){

} |

| Item Highlight | Not Applicable | **Property:** *itemHighlight*

 <ejs-maps id="maps"
(itemHighlight)="itemHighlight(\$event)"></ejs-maps>
 TS:
public itemHighlight(args){

} |

| Shape Highlight | Not Applicable | **Property:** *shapeHighlight*

 <ejs-maps id="maps"
(shapeHighlight)="shapeHighlight(\$event)"></ejs-maps>
 TS:
public
shapeHighlight(args){
} |

| Layer Rendering | Not Applicable | **Property:** *layerRendering*

 <ejs-maps id="maps"
(layerRendering)="layerRendering(\$event)"></ejs-maps>
 TS:
public
layerRendering(args){
} |

| Marker Rendering | Not Applicable | **Property:** *markerRendering*

 <ejs-maps id="maps"
(markerRendering)="markerRendering(\$event)"></ejs-maps>
 TS:
public
markerRendering(args){
} |

| Bubble Mouse Move | Not Applicable | **Property:** *bubbleMouseMove*

 <ejs-maps
id="maps" (bubbleMouseMove)="bubbleMouseMove(\$event)"></ejs-maps>
 TS:

public bubbleMouseMove(args){
} |

| Bubble Mouse Move | Not Applicable | **Property:** *annotationRendering*

 <ejs-maps
id="maps" (annotationRendering)="annotationRendering(\$event)"></ejs-maps>
 TS:

public annotationRendering(args){
} |

| Animation Complete | Not Applicable | **Property:** *animationComplete*

 <ejs-maps
id="maps" (animationComplete)="animationComplete(\$event)"></ejs-maps>
 TS:

public animationComplete(args){
} |

How To

Annotation in Angular Maps component

Annotations are used to mark the specific area of interest in the Maps with texts, shapes, or images. Any number of annotations can be added to the Maps component.

Initialize the Maps component with annotation option, text content or ID of an HTML element or an HTML string can be specified to render a new element that needs to be displayed in the Maps by using the [content](#) property. To specify the content position with [x](#) and [y](#) properties as mentioned in the following example.

[app.module.ts]

```

`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { AnnotationsService } from '@syncfusion/ej2-angular-maps';

@NgModule({
  imports: [ BrowserModule, MapsModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ],
  providers: [ AnnotationsService ]
})
export class AppModule { }
`

```

[app.component.ts]**APP.COMPONENT.TS**

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { africa_continent } from './africa-continent';
@Component({
  imports: [
    MapsModule
  ],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template: `<div class="control-section">
    <div align="center">
      <ejs-maps id="container" style="display:block;" [annotations]
= 'annotation'>
      <e-layers>
        <e-layer [shapeData]='shapeData' [shapeSettings]='shapeSettings'></e-
layer>
      </e-layers>
    </ejs-maps>
    </div>
  </div>
  <svg height="150" width="400">
    <defs>
      <linearGradient id="grad1" x1="0%" y1="0%" x2="0%" y2="100%">

```

```

        <stop offset="0%" style="stop-color:#C5494B;stop-
opacity:1"></stop>
        <stop offset="100%" style="stop-color:#4C134F;stop-
opacity:1"></stop>
    </linearGradient>
</defs>
</svg>
<div id="maps-annotation" style="display: none;">
    <div id="annotation">
        <div>
            <p style="margin-left:10px;font-size:13px;font-
weight:500">Facts about Africa</p>
        </div>
        <hr style="margin-top:-3px;margin-bottom:10px;border:0.5px solid
#DDDDDD">
        <div>
            <ul style="list-style-type:disc; margin-left:-20px;margin-
bottom:2px; font-weight:400">
                <li>Africa is the second largest and second most
populated continent in the world.</li>
                <li style="padding-top:5px;">Africa has 54 sovereign
states and 10 non-sovereign territories.
                </li>
                <li style="padding-top:5px;">Algeria is the largest
country in Africa, where as Mayotte is the smallest.</li>
            </ul>
        </div>
    </div>
</div>
<style>
    #annotation {
        color: #DDDDDD;
        font-size: 12px;
        font-family: Roboto;
        background: #3E464C;
        margin: 20px;
        padding: 10px;
        border-radius: 2px;
        width: 300px;
        box-shadow: 0px 2px 5px #666;
    }
</style>`,
    encapsulation: ViewEncapsulation.None
})
export class AppComponent {
    public shapeData = africa_continent;
    public shapeSettings = {
        fill: 'url(#grad1)'
    };
    public annotation:object[] = [
        {
            content: '#maps-annotation',
            x: '0%', y: '70%'
        }
    ];
}

```

MAIN.TS

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

Custom path in Angular Maps component

Maps component can be customized as the desired layout using the custom path map feature. Here, the Maps component has been showcased with normal geometry type shapes to represent the bus seat selection layout. Please refer to the following example to render the bus seat selection.

[app.module.ts]

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { MapsModule } from '@syncfusion/ej2-angular-maps';
import { SelectionService } from '@syncfusion/ej2-angular-maps';
@NgModule({
  imports: [ BrowserModule, MapsModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ],
  providers: [ SelectionService ]
})
export class AppModule { }
```

[app.component.ts]

```
`typescript
import { Component, ViewEncapsulation } from '@angular/core';
import { seatData } from './MapData/seatSelection-data/seatSelection';
@Component({
  selector: 'my-app',
  // specifies the template string for the maps component
  template: `<div class="control-section">
<div align='center'>
<ejs-maps id='container' height='400px' style="display:block;">
```

```

<e-layers>
<e-layer [shapeData]='seatdata' [selectionSettings]="selectionSettings" geometryType='Normal'></e-
layer>
</e-layers>
</ejs-maps>
</div>
</div>`,
encapsulation: ViewEncapsulation.None
})
export class AppComponent {
public seatdata=seatData;
public selectionSettings = {
enable: true,
opacity: 1,
enableMultiSelect: true
};
}
`

```

[index.html]

```

`html
<div class="col-lg-9 control-section">
<div style="width:200px;margin:auto;padding-bottom:20px">

<div style="padding-left:30px;font-size:20px;font-weight:400;">Bus seat selection</div>
</div>
<div style="border: 3px solid darkgray;width:200px;display:block;margin:auto;border-radius:5px">

<my-app>Loading AppComponent content here ...</my-app>
</div>
</div>
`

```

Drilldown in Angular Maps component

By clicking a continent, all the countries available in that continent can be viewed using the drill-down feature. For example, the countries in the **Africa** continent have been showcased here. To showcase all

the countries in **Africa** continent by clicking the [shapeSelected](#) event as mentioned in the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService, MapsTooltipService, HighlightService ,
SelectionService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation, ViewChild } from '@angular/core';
import { IShapeSelectedEventArgs, MapsComponent } from '@syncfusion/ej2-
angular-maps';
import { world_map } from './world-map';
import { africa_continent } from './africa-continent';
export interface ShapeData { continent?: string; }
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService, MapsTooltipService, HighlightService ,
SelectionService],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template: `<div class="control-section">
<div align="center">
<ejs-maps id="container" #drilldown
(shapeSelected)="shapeSelected($event)" style="display:block;">
<e-layers>
<e-layer [shapeData]='worldmap' layerType='Geometry'
shapePropertyPath='continent' shapeDataPath='continent'
[dataSource]='dataSource' [shapeSettings]='shapeSettings'
[markerSettings]='markerSettings'></e-layer>
<e-layer layerType='Geometry' [shapeData]='africa'
[shapeSettings]='africa_shapeSettings'
[highlightSettings]='highlightSettings'></e-layer>
</e-layers>
</ejs-maps>
</div>
</div>
<style>
.markerTemplate {
font-size: 12px;
color: white;
text-shadow: 0px 1px 1px black;
font-weight: 500
}
.markerTemplate {
height: 30px;
width: 30px;
display: block;
margin: auto;
}
</style>`,
  encapsulation: ViewEncapsulation.None
})
```

```

export class AppComponent {
  public worldmap = world_map;
  public africa = africa_continent;
  @ViewChild('drilldown')
  public maps?: MapsComponent;
  public shapeSelected = (args: IShapeSelectedEventArgs) : void => {
    let shape: string = (args.shapeData as ShapeData).continent as
string;
    if (shape === 'Africa') {
      (this.maps as MapsComponent).baseLayerIndex=1;
      this.maps?.refresh();
    }
  };
  public shapeSettings = {
    colorValuePath: 'drillColor'
  };
  public markerSettings=[{
    visible: true,
    template: '<div id="marker3" class="markerTemplate">Africa' +
      '</div>',
    dataSource: [
      { latitude: 10.97274101999902, longitude: 16.390625 }
    ],
    animationDuration: 0
  }];
  public africa_shapeSettings = {
    fill: '#80306A'
  };
  public highlightSettings = {
    enable: true,
    fill: '#80306A'
  };
  dataSource: any;
}

```

MAIN.TS

```

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

```

Marker type in Angular Maps component

Add different types of markers

Different marker objects can be added to the Maps component using the marker settings. To update different marker settings in Maps, please follow the given steps:

Step 1:

Initialize the Maps component with marker settings. Here, a marker has been added with specified latitude and longitude of California by using the [dataSource](#) property. To customize the shape of the marker using the [shape](#) property and change the border color and width of the marker using the [border](#) property as mentioned in the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template: `<ejs-maps id='container'>
    <e-layers>
    <e-layer [shapeData]='usmap' [markerSettings]='markerSettings'></e-
layer>
    </e-layers>
    </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public usmap=usa_map;
  public markerSettings = [
    {
      dataSource: [
        { latitude: 37.0000, longitude: -120.0000, city:
'California' }
      ],
      visible:true,
      shape:'Circle',
      fill:'white',
      width:3,
      animationDuration:0,
      border:{width:2,color:'#333'}
    }
  ]
}

```

MAIN.TS

```

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

```

Step 2:

Customize the above option for n number of markers as mentioned in the following example.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'

```

```

import { MarkerService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { usa_map } from './usa';
@Component({
  imports: [
    MapsModule
  ],
  providers: [MarkerService],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component.
  template: `<ejs-maps id='container'>
    <e-layers>
      <e-layer [shapeData]='usmap' [markerSettings]='markerSettings'></e-
layer>
    </e-layers>
  </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public usmap=usa_map;
  // Initializing Map with Marker sttings.
  public markerSettings = [
    {
      dataSource: [
        { latitude: 37.0000, longitude: -120.0000, city:
'California' }
      ],
      visible:true,
      shape:'Circle',
      fill:'white',
      width:3,
      animationDuration:0,
      border:{width:2,color:'#333'}
    },
    {
      dataSource: [
        { latitude: 40.7127, longitude: -74.0059, city: 'New York' }
      ],
      visible:true,
      shape:'Rectangle',
      fill:'yellow',
      width:15,
      height:4,
      animationDuration:0,
      border:{width:2,color:'#333'}
    },
    {
      dataSource: [
        { latitude: 42, longitude: -93, city: 'Iowa' }
      ],
      visible:true,
      shape:'Diamond',
      fill:'white',
      width:10,
      height:10,
      animationDuration:0,

```



```

        border:{width:2,color:'blue'}
      },
      {
        dataSource: [
          { latitude:36.499589049395055, longitude:-
103.042108197135548, city:'New Mexico'}
        ],
        visible:true,
        shape:'Balloon',
        fill:'red',
        width:10,
        height:15,
        animationDuration:0,
        border:{width:2,color:'#333'}
      },
      {
        dataSource: [
          { latitude:36.360624205142919, longitude:-
94.595916790727287, city:'Oklahoma'}
        ],
        visible:true,
        shape:'Triangle',
        fill:'blue',
        width:10,
        animationDuration:0,
        border:{width:2,color:'#333'}
      }
    ]
  }
}

```

MAIN.TS

```

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

```

Multiple layer in Angular Maps component

The multilayer support allows loading multiple shape files in a single container and enables Maps to display more information. The shape layer is the main layer of the Maps. Multiple layers can be added in a shape layer as **SubLayer** using the [type](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { usa_map } from './usa';
import { california } from './california';
import { texas } from './texas';
@Component({
  imports: [
    MapsModule
  ],

```

```

standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component.
  template: `<div class="control-section">
    <div align="center">
      <ejs-maps id='container' style="display:block;">
        <e-layers>
          <e-layer [shapeData]='usMap' [shapeSettings]='us_shapeSettings'></e-
layer>
          <e-layer [shapeData]='texas' type='SubLayer'
[shapeSettings]='texas_shapeSettings'></e-layer>
          <e-layer [shapeData]='california' type='SubLayer'
[shapeSettings]='california_shapeSettings'></e-layer>
        </e-layers>
      </ejs-maps>
    </div>
  </div>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public usMap = usa_map;
  public us_shapeSettings = {
    fill: '#E5E5E5',
    border: {
      color: 'black',
      width: 0.1
    }
  };
  public texas = texas;
  public texas_shapeSettings = {
    fill: 'rgba(141, 206, 255, 0.6)',
    border: {
      color: '#1a9cff',
      width: 0.25
    }
  };
  public california=california;
  public california_shapeSettings = {
    fill: 'rgba(141, 206, 255, 0.6)',
    border: {
      color: '#1a9cff',
      width: 0.25
    }
  }
}
}

```

MAIN.TS

```

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

```

Zooming in Angular Maps component

The center position zooming can be achieved by using the [centerPosition](#) and [zoomFactor](#) properties as mentioned in the following example. The center position is used to configure the zoom level of Maps, and the zoom factor is used to specify the center position where the Maps should be displayed.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MapsModule } from '@syncfusion/ej2-angular-maps'
import { ZoomService } from '@syncfusion/ej2-angular-maps'
import { Component, ViewEncapsulation } from '@angular/core';
import { world_map } from './world-map';
@Component({
  imports: [
    MapsModule
  ],
  providers: [ZoomService],
  standalone: true,
  selector: 'app-container',
  // specifies the template string for the maps component
  template: `<ejs-maps id='rn-container' [zoomSettings]='zoomSettings'
[centerPosition]='centerPosition' style="display:block;">
    <e-layers>
    <e-layer [shapeData]='worldmap'></e-layer>
    </e-layers>
  </ejs-maps>`,
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
  public worldmap:object = world_map;
  public zoomSettings:object = {
    enable:true,
    zoomFactor:13,
    maxZoom: 25
  };
  public centerPosition:object = {
    latitude: 25.54244147012483,
    longitude: -89.62646484375
  }
}
```

MAIN.TS

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

MaskedTextBox

Getting started with Angular Maskedtextbox component

The following section explains the steps required to create the MaskedTextBox component and also it demonstrates the basic usage of the MaskedTextBox.

Dependencies

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

```
`javascript
|-- @syncfusion/ej2-angular-inputs
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-angular-popups
|-- @syncfusion/ej2-angular-buttons
|-- @syncfusion/ej2-angular-splitbuttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-splitbuttons
`,`
```

Setup angular environment

Angular provides the easiest way to set angular CLI projects using [Angular CLI](#) tool.

Install the CLI application globally to your machine.

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

Create a new application

```
`bash
ng new syncfusion-angular-maskedtextbox
`,`
```

By default, it install the CSS style base application. To setup with SCSS, pass `--style=scss` argument on create project.

Example code snippet.

```
`bash
ng new syncfusion-angular-maskedtextbox --style=scss
`,`
```

Navigate to the created project folder.

```
`bash
```

```
cd syncfusion-angular-maskedtextbox
```

```
,
```

Installing Syncfusion MaskedTextBox Package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-inputs:"20.2.38-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 MaskedTextBox module

Import MaskedTextBox module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-inputs`.

```
`javascript
```

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// imports the MaskedTextBoxModule for the MaskedTextBox component
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs';
import { AppComponent } from './app.component';
@NgModule({
  // declaration of ej2-angular-inputs module into NgModule
  imports: [ BrowserModule, MaskedTextBoxModule ],
  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-inputs/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-inputs/styles/material.css';
`

```

Adding MaskedTextBox component

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

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

```

`javascript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: <ejs-maskedtextbox></ejs-maskedtextbox>
})

```

```
export class AppComponent { }
,
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Set the mask

You can set the mask to the MaskedTextBox to validate the user input by using the [mask](#) property.

For more information about the usage of mask and configuration, refer to this [link](#).

The following example demonstrates the usage of mask element **0** that allows any single digit from **0** to **9**.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // sets mask format to the MaskedTextBox
  template: `
    <ejs-maskedtextbox mask='000-000-0000'></ejs-maskedtextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
,
```

Running the application

After completing the configuration required to render a basic MaskedTextBox, run the following command to display the output in your default browser.

```
ng serve
,
```

The following example illustrates the output in your browser.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
```

```
imports: [
    MaskedTextBoxModule, FormsModule
],
standalone: true,
selector: 'app-root',
// sets mask format to the MaskedTextBox
template: `<label class='label'>Enter the mobile number</label>`+
    `<ejs-maskedtextbox mask='000-000-0000'></ejs-maskedtextbox>`
})
export class AppComponent {
    constructor() {
    }
}
```

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 MaskedTextBox, the `value` property supports two-way binding functionality.

The following example demonstrates two-way binding functionality with the MaskedTextBox and HTML input element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
    imports: [
        MaskedTextBoxModule, FormsModule
    ],
    standalone: true,
    selector: 'app-root',
    // input element for checking the two-way binding support using value
    // property
    template: `
        <div class='e-input-group' style='margin-bottom:30px'>
            <ejs-input class='e-input' type='text' [(ngModel)]='value'
placeholder='Mobile Number' />
        </div>
        <ejs-maskedtextbox mask='000-000-0000' [(value)]='value'></ejs-
maskedtextbox>
    `
})
export class AppComponent {
    public value: any;
    constructor() {
    }
}
```


MAIN.TS

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

Reactive form

MaskedTextBox is a form component and validation is playing vital role in forms to get the valid data.

Here to showcase the MaskedTextBox with form validations we have used the reactive form.

For more details about Reactive Forms refer: <https://angular.io/guide/reactive-forms>.

- To use reactive forms, import **ReactiveFormsModule** from the **@angular/forms** package and add it to your NgModule's imports array. In addition to this, **FormGroup**, **FormControl** should be imported to the app component.
- The **FormGroup** is used to declare **formGroupName** for the form. The constructor of this **FormGroup** then takes an object, that can contain sub-form-groups and **FormControls**.
- The **FormControl** is used to declare **formControlName** for form controls.

The following example demonstrates how to use the reactive forms.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject } from '@angular/core';
import { MaskedTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MaskedTextBoxModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: './template.html',
})
export class AppComponent {
  skillForm?: FormGroup | any;
  build: FormBuilder;
  constructor(@Inject(FormBuilder) private builder: FormBuilder) {
    this.build = builder;
    this.createForm();
  }
  createForm() {
    this.skillForm = this.build.group({
      mask: ['', Validators.required],
      username: ['', Validators.required],
    });
  }
}
```

```

    }
    get username() { return this.skillForm?.get('username'); }
    get mask() { return this.skillForm?.get('mask'); }
    onSubmit() {
        alert("You have entered the value: " + this.mask?.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

- [How to perform custom validation using FormValidator](#)
- [How to customize the UI appearance of the control](#)
- [How to set cursor position while focus on the input textbox](#)
- [How to display numeric keypad when focus on mobile devices](#)

Mask configuration in Angular Maskedtextbox component

The mask is a combination of standard and custom mask elements that validates the user input based on its behavior.

When the mask value is empty, the MaskedTextBox behaves as an input element with text type.

Standard mask elements

The following table shows the list of mask elements and its behavior based on [MSDN](#) standard.

The mask can be formed by combining any one or more of these mask elements.

Mask Element	Description
-----	-----
0	Digit required. This element will accept any single digit from 0 to 9 .
9	Digit or space, optional.
#	Digit or space, optional, Plus(+) and minus(-) signs are allowed.
L	Letter required. It will accept letters a-z and A-Z .
?	Letter or space, optional.
&	Requires a character.
C	Character or space, optional.
A	Alphanumeric (A-Za-z0-9) required.
a	Alphanumeric (A-Za-z0-9) or space, optional.
<	Shift down. Converts all characters to lower case.

| > | Shift up. Converts all characters to upper case. |

| | | Disable a previous shift up or shift down. |

| \\\ | Escapes a mask character, turning it into a literal. |

| All other characters | Literals. All non-mask elements (literals) will appear as themselves within MaskedTextBox. |

The following example demonstrates the usage of standard mask elements.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule, FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets different types of standard masks to the MaskedTextBox component
  template: `
    <div class='wrap'>
      <ejs-maskedtextbox mask='#####' placeholder='Mask #####'
(ex: 012+-) floatLabelType= 'Always'></ejs-maskedtextbox>
    </div>
    <div class='wrap'>
      <ejs-maskedtextbox mask='LLLLLL' placeholder='Mask LLLLLL'
(ex: Sample) floatLabelType= 'Always'></ejs-maskedtextbox>
    </div>
    <div class='wrap'>
      <ejs-maskedtextbox mask='&&&&' placeholder='Mask &&&&'
(ex: A12@#) floatLabelType= 'Always'></ejs-maskedtextbox>
    </div>
    <div class='wrap'>
      <ejs-maskedtextbox mask='>LLL<LLL' placeholder='Mask
>LLL<LL (ex: SAMple) floatLabelType= 'Always'></ejs-maskedtextbox>
    </div>
    <div class='wrap'>
      <ejs-maskedtextbox mask='\\A999' placeholder='Mask \\A999'
(ex: A321) floatLabelType= 'Always'></ejs-maskedtextbox>
    </div>`
  })
export class AppComponent {
  constructor() {
  }
}
```

MAIN.TS

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

Custom mask elements

Other than the above standard mask elements, the mask can be configured with the custom characters or regular expression to define a custom behavior.

Custom characters

You can define any of the non-mask element as the mask element and its behavior through the [customCharacters](#) property.

In the following example, non-mask element **P** accepts the values **P, A, p, a** and **M** accepts the values **M, m** as mentioned in the custom characters collection.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule, FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets mask format to the MaskedTextBox
  template: `
    <ejs-maskedtextbox [customCharacters]="customMaskChar"
    mask='00:00' >PM' placeholder='Time (ex: 10:00 PM, 10:00 AM)' floatLabelType=
    'Always'></ejs-maskedtextbox>
  `
})
export class AppComponent {
  // sets custom characters collection for non-mask elements 'P' and 'M'
  public customMaskChar: Object = { P: 'P,A,p,a', M: 'M,m' };
  constructor() {
  }
}
```

MAIN.TS

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

Regular expression

Instead of the mask element, you can define your own regular expression to validate the input of a particular input place.

The regular expressions should be wrapped by the square brackets (e.g., **[Regex]**).

In the following example, regular expression has been set for each input places.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets set of regular expression for each input place as mask
  template: `
    <ejs-maskedtextbox name="mask_value" #mask="" mask='[0-2][0-9][0-9].[0-2][0-9][0-9].[0-2][0-9][0-9].[0-2][0-9][0-9]' placeholder='IP Address (ex: 212.212.111.222)' floatLabelType= 'Always'></ejs-maskedtextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

MAIN.TS

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

Prompt character

The Prompt character is a prompting symbol in the MaskedTextBox for the mask elements. The symbol is used to show the input positions in the MaskedTextBox.

You can customize the prompt character of MaskedTextBox by using the [promptChar](#) property.

The following example demonstrates the MaskedTextBox with customized prompt character as `#`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule, FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets the prompting symbol to the MaskedTextBox
  // sets mask format to the MaskedTextBox
  template: `
    <ejs-maskedtextbox promptChar="#" mask='999-999-9999'></ejs-maskedtextbox>
  `
})
```

```

  })
  export class AppComponent {
    constructor() {
    }
  }
}

```

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 Maskedtextbox component

The Maskedtextbox 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 Maskedtextbox 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 MaskedTextBox is characterized with complete ARIA Accessibility support that helps to access through the on-screen readers and other assistive technology devices. This component is designed with the reference of the guidelines document given in [WAI ARAI Accessibility practices](#).

The MaskedTextBox uses the `textbox` role and following ARIA properties for its element based on its state.

| Property | Functionality |

| --- | --- |

| aria-live | The `aria-live` attribute indicates the priority of updates to a live region. |

| aria-disabled | The `aria-disabled` property indicates the disabled state of the MaskedTextBox. |

| aria-valuenow | The `aria-valuenow` property specifies the current value of the MaskedTextBox. |

| aria-invalid | The `aria-invalid` property indicates that the user input is incorrect or not within the acceptable ranges. |

| aria-placeholder | The `aria-placeholder` is a short hint to help the users with data entry when the MaskedTextBox has no value. |

| aria-labelledby | The `aria-labelledby` property indicates the floating label element of the MaskedTextBox. |

Ensuring accessibility

The MaskedTextBox component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the MaskedTextBox component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the MaskedTextBox component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Style appearance in Angular Maskedtextbox component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the appearance of MaskedTextBox wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
`css
/ To specify height, font size, and border /

.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input, .e-input-group textarea.e-
input, .e-input-group.e-control-wrapper textarea.e-input {
font-size: 20px;
border-color: red;
height: 40px;
border: 2px solid;
}
`
```

Customizing the MaskedTextBox element on hovering

Use the following CSS to customize the Input Mask element on hovering

```
`css
/ To specify border /

.e-input-group input.e-input, .e-input-group input.e-input:hover:not(.e-success):not(.e-warning):not(.e-
error):not([disabled]):not(:focus), .e-input-group.e-control-wrapper input.e-input,.e-input-group.e-
control-wrapper input.e-input:hover:not(.e-success):not(.e-warning):not(.e-
error):not([disabled]):no(:focus){
border: 3px solid red;
}
`
```

How To

Perform custom validation using form validator in Angular Maskedtextbox component

To perform custom validation on the MaskedTextBox use the FormValidator along with custom validation rules.

In the following example, the MaskedTextBox is validated for invalid mobile number by adding custom validation in the rules collection of the FormValidator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { FormsModule } from '@angular/forms'
import { Component, ViewChild } from '@angular/core';
import { MaskedTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
@Component({
imports: [
```



```

    MaskedTextBoxModule, FormsModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets mask format to the MaskedTextBox
  template: `
    <form #formElement class="form-horizontal">
      <div class="form-group">
        <div class="col-sm-6">
          <br/><ejs-maskedtextbox id="masktextbox" #mask="" mask='000-
000-0000' name="mask_value" placeholder= 'Mobile Number' floatLabelType=
'Always'></ejs-maskedtextbox>
        </div>
        <button type="button" id="submit_btn" (click)="btnClick()"
style="margin-top: 10px">Submit</button>
      </div>
    </form>`
  })
export class AppComponent {
  @ViewChild('formElement') element?: any;
  @ViewChild('mask') mask?: MaskedTextBoxComponent;
  public formObject?: FormValidator;
  ngAfterViewInit() {
    let customFn: (args: { [key: string]: string }) => boolean = (args: {
[key: string]: string }) => {
      let argsLength = (args as
any).element.ej2_instances[0].value.length;
      if(argsLength != 0) {
        return argsLength >= 10;
      }
      else {
        return true;
      }
    };
    let custom: (args: { [key: string]: string }) => boolean = (args: {
[key: string]: string }) => {
      let argsLength = (args as
any).element.ej2_instances[0].value.length;
      if(argsLength == 0) {
        return 0;
      }
      else {
        return argsLength;
      }
    };
    // sets required property in the FormValidator rules collection
    let options: FormValidatorModel = {
      rules: {
        'mask_value': { numberValue: [customFn, 'Enter valid mobile
number'] },
      }
    };
    this.formObject = new FormValidator(this.element.nativeElement,
options);
    //FormValidator rule is added for empty MaskedTextBox
    this.formObject.addRules('mask_value', { maxLength: [custom, 'Enter
mobile number'] });
  }
}

```

```
// places error label outside the MaskedTextBox using the
customPlacement event of FormValidator
let customPlace: (element: HTMLElement, error: HTMLElement) => void
= (element: HTMLElement, error: HTMLElement) => {
    (document.querySelector(".form-group") as
Element).appendChild(error);
};
this.formObject.customPlacement = customPlace;
}
public btnClick(): void {
    // validates the MaskedTextBox
    this.formObject?.validate("mask_value");
    // checks for incomplete value and alerts the formt submit
    if (this.mask?.element.value !== "" &&
this.mask?.element.value.indexOf(this.mask.promptChar) === -1) {
        alert("Submitted");
    }
}
}
```

MAIN.TS

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

Set cursor position while focus on the input textbox in Angular Maskedtextbox component

By default, on focusing the MaskedTextBox the entire mask gets selected. You can customize by using any one of the following methods:

- Setting cursor position at the start of the MaskedTextBox.
- Setting cursor position at the end of the MaskedTextBox.
- Setting cursor at the specified position in the MaskedTextBox.

The **selectionStart** and **selectionEnd** set to **0** instead of the input element value's length, when we focus on a MaskedTextBox control filled with all mask characters. This is the default behavior of the HTML 5 input element.

Following is an example that demonstrates the above cases to set cursor position in the MaskedTextBox using [focus](#) event.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
```

```
// sets mask format to the MaskedTextBox
template: `
    <div class="col-sm-6">
        <br/><ejs-maskedtextbox #mask="" id="mask1" mask='00000-
00000' value='93828-32132' name="mask_value1" placeholder= 'Default cursor
position' floatLabelType= 'Always'></ejs-maskedtextbox><br/>
        <ejs-maskedtextbox #mask="" id="mask2" mask='00000-00000'
value='83929-4342' name="mask_value2" placeholder= 'Cursor positioned at
start' floatLabelType= 'Always' (focus)= "onStartfocus($event)"></ejs-
maskedtextbox><br/>
        <ejs-maskedtextbox #mask="" id="mask3" mask='00000-00000'
value='83929-3213' name="mask_value3" placeholder= 'Cursor positioned at
end' floatLabelType= 'Always' (focus)= "onEndfocus($event)"></ejs-
maskedtextbox><br/>
        <ejs-maskedtextbox #mask="" id="mask4" mask='+1 000-000-
0000' value='234-432-432' name="mask_value4" placeholder= 'Cursor at
specified position' floatLabelType= 'Always' (focus)=
"onSpecificfocus($event)"></ejs-maskedtextbox>
    </div>
`
}))
export class AppComponent {
    public onStartfocus(args: any): void {
        //sets cursor position at start of MaskedTextBox
        args.selectionEnd= args.selectionStart = 0;
    }
    public onEndfocus(args: any): void {
        //sets cursor position at end of MaskedTextBox
        args.selectionStart=args.selectionEnd = args.maskedValue.length;
    }
    public onSpecificfocus(args: any): void {
        //sets cursor at specified position
        args.selectionStart = 3;
        args.selectionEnd = 3;
    }
}
```

MAIN.TS

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

Display numeric keypad when focus on mobile devices in Angular Maskedtextbox component

By default, on focusing the MaskedTextBox, alphanumeric keypad will be displayed on mobile devices. Sometimes only numeric keypad for number values is needed, and this can be achieved by setting "type" property to tel.

Refer to the following example to enable numeric keypad in MaskedTextBox.

APP.COMPONENT.TS

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

```
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets mask format to the MaskedTextBox
  template: `
    <div class="col-sm-6">
      <br/><ejs-maskedtextbox #mask="" mask='999-99999' value=
"342-45432" name="mask_value" type="tel"></ejs-maskedtextbox>
    </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));
```

Customize the ui appearance of the control in Angular Maskedtextbox component

The appearance of the MaskedTextBox can be changed by adding custom `cssClass` to the component and enabling styles.

Refer to the following example to change the appearance of the MaskedTextBox.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MaskedTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
@Component({
  imports: [
    MaskedTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // sets mask format to the MaskedTextBox
  template: `
    <div class="col-sm-6">
      <br/><ejs-maskedtextbox #mask="" id="mask1" mask='00000'
value= "42648" name="mask_value" cssClass="e-style" placeholder= 'Enter user
ID' floatLabelType= 'Always' (focus)= "onfocus($event)"></ejs-maskedtextbox>
    </div>
  `
})
export class AppComponent {
  public onfocus(args: any): void {
    //sets cursor position at start of MaskedTextBox
  }
}
```

```

        args.selectionEnd= args.selectionStart;
    }
}

```

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 Maskedtextbox component

This article describes the API migration process of MaskEdit component from Essential JS 1 to Essential JS 2.

Common

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Adding custom class | **Property:** *cssClass*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="9999" cssClass = "custom" /> | **Property:**
cssClass

<ejs-maskedtextbox #mask cssClass = "custom" mask='9999'></ejs-
maskedtextbox> |

| Destroy editor | Not Applicable | **Method:** *destroy*

<ejs-maskedtextbox #mask
mask="00-000"></ejs-maskedtextbox>
@ViewChild("mask") mask:
MaskedTextBoxComponent;
mask.destroy(); |

| Disable the maskedit control | **Method:** *disable*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000" value="1234" />
var maskObj =
\$("#mask").data("ejMaskEdit");
maskObj.disable(); | **Can be achieved using**

<ejs-
maskedtextbox #mask mask="00-00" value="1234"></ejs-maskedtextbox>
@ViewChild("mask") mask: MaskedTextBoxComponent;
mask.enabled = false; |

| Enable the maskedit control | **Method:** *enable*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000" value="1234" />
var maskObj =
\$("#mask").data("ejMaskEdit");
maskObj.enable(); | **Can be achieved using**

<ejs-
maskedtextbox #mask mask="00-00" value="1234"></ejs-maskedtextbox>
@ViewChild("mask") mask: MaskedTextBoxComponent;
mask.enabled=true; |

| Control state | **Property:** *enabled*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="00-000" enabled="false" /> | **Property:**
enabled

<ejs-maskedtextbox mask='00-000' enabled=false></ejs-maskedtextbox> |

| Persistence | **Property:** *enablePersistence*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000" enablePersistence="true" /> | **Property:**
enablePersistence

<ejs-maskedtextbox mask='0000' enablePersistence=true></ejs-
maskedtextbox> |

| Triggers when editor is focused in | **Event:** *focusIn*

inputMode="ej.InputMode.Text" maskFormat="00-00" (focusIn)="focusIn()"/>
</input> | **Event:** *focus*
<ej-maskedtextbox mask='0000' (focus) =
"onFocus()"></ej-maskedtextbox> | **Script:**
public onFocus(): void {} |

| Triggers when editor is focused out | **Event:** *focusOut*

inputMode="ej.InputMode.Text" maskFormat="0000" (focusOut)="focusOut()"/>
</input> | **Event:** *blur*
<ej-maskedtextbox mask='0000' (blur) =
"onBlur()"></ej-maskedtextbox> | **Script:**
public onBlur(): void {} |

| Sets height | **Property:** *height*

maskFormat="0000" height="30px" /> | **Can be achieved using,**
<ej-maskedtextbox
#mask cssClass = "custom" mask='0000'></ej-maskedtextbox> | **Css:**
.e-maskedtextbox.custom { height: 30px; } |

| HTML Attributes | **Property:** *htmlAttributes*

inputMode="ej.InputMode.Text" maskFormat="0000" [htmlAttributes]= "htmlAttr" />
</input> | **Can be achieved using,**
<ej-maskedtextbox mask='999-99' value='12345' name="masked" ></ej-maskedtextbox> |

| Specifies Input mode | **Property:** *inputMode*

inputMode="ej.InputMode.Text" maskFormat="0000" /> | **Can be achieved using,**
<ej-maskedtextbox mask='999-99' value='12345' type="password" ></ej-maskedtextbox> |

| Triggers on key press | **Event:** *keyPress*

inputMode="ej.InputMode.Text" maskFormat="0000" (keyPress)="onKeyPress()"/>
</input> | **Can be achieved using**
<ej-maskedtextbox mask='999-99'
id="mask" value='12345' placeholder= "Masked TextBox" floatLabelType= 'Always'
(created)="onCreate(\$event)"></ej-maskedtextbox> | **Script:**
public onCreate()
{
document.getElementById("mask").addEventListener('keypress', (event:any)=>{
/> } |

| Triggers on key up | **Event:** *keyUp*

inputMode="ej.InputMode.Text" maskFormat="0000" (keyUp)="onKeyUp()"/>
</input> | **Can be achieved using**
<ej-maskedtextbox mask='999-99'
id="mask" value='12345' placeholder= "Masked TextBox" floatLabelType= 'Always'
(created)="onCreate(\$event)"></ej-maskedtextbox> | **Script:**
public onCreate()
{
document.getElementById("mask").addEventListener('keyup', (event:any)=>{
/> } |

| Triggers on mouse out in maskedit control | **Event:** *mouseOut*

inputMode="ej.InputMode.Text" maskFormat="0000-000" (mouseOut)="onMouseOut()"/>
</input> | **Can be achieved using**
<ej-maskedtextbox mask='999-99'
id="mask" value='12345' placeholder= "Masked TextBox" floatLabelType= 'Always'
(created)="onCreate(\$event)"></ej-maskedtextbox> | **Script:**
public onCreate()
{
document.getElementById("mask").addEventListener('mouseout', (event:any)=>{
/> } |

| Triggers when mouse over in maskedit control | **Event:** *mouseOver*

Can be achieved using
`<ejs-maskedtextbox mask='999-99' id='mask' value='12345' placeholder='Masked TextBox' floatLabelType='Always' (created)="onCreate($event)"></ejs-maskedtextbox>`
Script

```
public onCreate() {
  document.getElementById("mask").addEventListener('mouseover', (event: any) => {});
}
```

| Name of maskedit control | **Property:** *name*

Can be achieved using
`<ejs-maskedtextbox mask='999-99' value='12345' name='masked'></ejs-maskedtextbox>`

| Triggers on keydown | **Event:** *onKeyDown*

Can be achieved using
`<ejs-maskedtextbox mask='999-99' id='mask' value='12345' placeholder='Masked TextBox' floatLabelType='Always' (created)="onCreate($event)"></ejs-maskedtextbox>`
Script

```
public onCreate() {
  document.getElementById("mask").addEventListener('keydown', (event: any) => {});
}
```

| Read only | **Property:** *readOnly*

Can be achieved using
`<ejs-maskedtextbox #mask width="100px" mask='0000' enabled = false></ejs-maskedtextbox>`
CSS

```
.e-mask {
  background-image: none !important;
  border-bottom-color: rgba(0, 0, 0, 0.42) !important;
}
```

| Right to left | **Property:** *textAlign*

Property: *enableRtl*
`<ejs-maskedtextbox #mask enableRtl=true mask='9999'></ejs-maskedtextbox>`

| Sets width | **Property:** *width*

Property: *width*
`<ejs-maskedtextbox #mask width="100px" mask='0000'></ejs-maskedtextbox>`

Mask Configuration

<!-- markdownlint-disable MD033 -->

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Triggers on value change | **Event** *change*

Script

```
onChange() {}
```

Event: *change*
`<ejs-maskedtextbox mask='00-00' (change) = "onChange()"></ejs-maskedtextbox>`
Script

```
public onChange(): void {}
```

| Clears maskedit text/value | **Method:** *clear*

inputMode="ej.InputMode.Text" maskFormat="0000" value="1234"/>
var maskObj =
\$("#mask").data("ejMaskEdit");
maskObj.clear(); | **Can be achieved using**
<script>
<ejs-maskedtextbox #mask mask="00-00" value="1234"></ejs-maskedtextbox>
</script>
@ViewChild("mask") mask: MaskedTextBoxComponent;
mask.value = "" |

| Triggers on creation | **Event:** *create*

inputMode="ej.InputMode.Text" maskFormat="00-00" (create)="onCreate()"/>
<script>
onCreate() {} | **Event:** *created*
<ejs-maskedtextbox mask='00-00'
(created) = "onCreated()"/>
<script>
public onCreated(): void
{} |

| Custom Character | **Property:** *customCharacter*

inputMode="ej.InputMode.Text" maskFormat="C-0000" customCharacter="#"> | **Property:**
customCharacters
<ejs-maskedtextbox mask='C-0000'
[customCharacters]="customCharacters"></ejs-maskedtextbox>
<script>
public
customCharacters: object = {C: '#'}; |

| Triggers when maskedit control is destroyed | **Event:** *destroy*

inputMode="ej.InputMode.Text" maskFormat="00-00" (destroy)="onDestroy()"/>
<script>
onDestroy() {} | **Event:** *destroyed*
<ejs-maskedtextbox mask='00-00'
(destroyed) = "onDestroyed()"/>
<script>
public
onDestroyed(): void {} |

| Placeholder float type | Not Applicable | **Property:** *floatLabelType*
<ejs-maskedtextbox
mask='9,999' placeholder="Enter value" floatLabelType="Auto"></ejs-maskedtextbox> |

| Gets pure value of maskedit control | **Method:** *getStrippedValue*

inputMode="ej.InputMode.Text" maskFormat="0000" value="123"/>
<script>
var
maskObj = \$("#mask").data("ejMaskEdit");
maskObj.getStrippedValue(); | **Can be achieved
using**
<script>
<ejs-maskedtextbox #mask mask="00-00" value="1234"></ejs-
maskedtextbox>
@ViewChild("mask") mask: MaskedTextBoxComponent;
alert(mask.value); |

| Get whole maskedit value | **Method:** *getUnstrippedValue*

inputMode="ej.InputMode.Text" maskFormat="00-00" value="1234"/>
var maskObj =
\$("#mask").data("ejMaskEdit");
maskObj.getUnstrippedValue(); | **Method:** *getMaskedValue*
<script>
<ejs-maskedtextbox #mask mask="00-00" value="1234"></ejs-
maskedtextbox>
@ViewChild("mask") mask: MaskedTextBoxComponent;
mask.getMaskedValue(); |

| Hides prompt character on focus out | **Property:** *hidePromptOnLeave*

inputMode="ej.InputMode.Text" maskFormat="aaaa" hidePromptOnLeave="true"/> | **Can be
achieved using**
<ejs-maskedtextbox mask='999-99' #mask id="mask1" value='12345'
focus="onFocus" placeholder="Masked TextBox" floatLabelType= 'Always'
(created)="onCreate(\$event)"></ejs-maskedtextbox>
<script>
@ViewChild("mask")
mask: MaskedTextBoxComponent; public


```
onCreate(){<br/>document.getElementById("mask1").addEventListener('blur',(event:any)=>{<br/>mask.promptChar= " "<br/>});<br/>}<br/>public onFocus(){<br/>mask.promptChar="_"<br/>}
```

| Mask format | **Property:** *maskFormat*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="99,999"/> | **Property:** *mask*

<ejs-
maskedtextbox mask='99,999'></ejs-maskedtextbox> |

| Prompt character | Not Applicable | **Property:** *promptChar*

<ejs-maskedtextbox
mask='0000' promptChar="#"></ejs-maskedtextbox> |

| Clear Button | Not Applicable | **Property:** *showClearButton*

<ejs-maskedtextbox
mask='aaaa' showClearButton=true></ejs-maskedtextbox> |

| Prompt character display | **Property:** *showPromptChar*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="\$99-999" showPromptChar="false"/> | **Can be
achieved using**
<ejs-maskedtextbox mask='0000' promptChar=" "></ejs-maskedtextbox> |

| Show rounded corner | **Property:** *showRoundedCorner*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000" showRoundedCorner="true"/> | **Can be
achieved using**
<ejs-maskedtextbox mask="000-000-000" id="mask1"
floatLabelType="Always" cssClass="e-style"></ejs-
maskedtextbox>
Css
#mask1{
border: 2px solid grey;
padding:
7px;
border-radius: 10px;
}
.e-control-wrapper.e-mask.e-float-input.e-style .e-
float-line::before ,.e-control-wrapper.e-mask.e-float-input.e-style .e-float-
line::after
{
background: none ;
}

| Value of maskedit control | **Property:** *value*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000" value="1234"/> | **Property:** *value*

<ejs-maskedtextbox mask='0000' value="1234"></ejs-maskedtextbox> |

| Displays hint on maskedit control | **Property:** *watermarkText*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="9999" watermarkText="Enter value"/> |
Property: *placeholder*

<ejs-maskedtextbox mask='9999' placeholder="Enter
value"></ejs-maskedtextbox> |

Validation

```
<!-- markdownlint-disable MD033 -->
```

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Displays error until correct value is entered | **Property:** *showError*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="99-999" showError="true"/> | **MaskedTextBox
by default shows error until the correct value is entered**
<script
<ejs-maskedtextbox #mask
mask="00-00" value="1234"></ejs-maskedtextbox> |

| Validation message | **Property:** *validationMessage*

<input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="0000"/>

<script
constructor()
{
this.validationRules = {required: true};
this.validationMessage = {required:
"Required value"}
} | **Can be achieved using Form Validation**
<form #formElement

```

class="form-horizontal"><br /><ejs-maskedtextbox id="masktextbox" #mask="" mask='000-000-0000' name="maskvalue" placeholder= 'Mobile Number' floatLabelType= 'Always'><br /></ejs-maskedtextbox><br /></form><br /><script><br /> @ViewChild('formElement') element;
<br />@ViewChild('mask') mask: MaskedTextBoxComponent;</br>public formObject:
FormValidator;<br /> let options: FormValidatorModel = {<br />rules: {<br />'maskvalue': {
required: [true, 'Enter valid mobile number'] },<br />}<br />}<br />this.formObject = new
FormValidator(this.element.nativeElement, options);<br />let customPlace: (element:
HTMLElement, error: HTMLElement) => void = (element: HTMLElement, error: HTMLElement)
=> {<br />document.querySelector(".form-
group").appendChild(error);<br />}<br />this.formObject.customPlacement = customPlace;<br />
|
| Validation Rules | Property: validationRules<br /><br /><input ej-maskedit
inputMode="ej.InputMode.Text" maskFormat="00-00"/><br /><br /><script><br />constructor()
{<br />this.validationRules = {required: true};<br />} | Can be achieved using Form
Validation<br /><form #formElement class="form-horizontal"><br /><ejs-maskedtextbox
id="masktextbox" #mask="" mask='000-000-0000' name="maskvalue" placeholder= 'Mobile
Number' floatLabelType= 'Always'><br /></ejs-maskedtextbox><br /></form><br /><script><br />
@ViewChild('formElement') element; <br /> @ViewChild('mask') mask:
MaskedTextBoxComponent;</br>public formObject: FormValidator;<br /> let options:
FormValidatorModel = {<br />rules: {<br />'maskvalue': { required: [true]
},<br />}<br />}<br />this.formObject = new FormValidator(this.element.nativeElement,
options);<br />let customPlace: (element: HTMLElement, error: HTMLElement) => void =
(element: HTMLElement, error: HTMLElement) => {<br />document.querySelector(".form-
group").appendChild(error);<br />}<br />this.formObject.customPlacement = customPlace; |

```

Mention

Getting started with Angular Mention component

This section explains how to create a simple [Link to the Video](#) component and demonstrates the basic usage of the Mention component in an Angular environment.

To get started quickly with angular Mention component using angular CLI, you can check the video below.

Dependencies

The list of dependencies required to use the Mention component in your application is given below:

```

`javascript
|-- @syncfusion/ej2-angular-dropdowns
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-lists

```

```
|-- @syncfusion/ej2-popups  
,
```

Setup Angular environment

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

```
,
```

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

```
,
```

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
,
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion Mention package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-dropdowns --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-dropdowns@ngcc](#) package to the application.

```
`bash
```

```
npm install @syncfusion/ej2-angular-dropdowns@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-dropdowns:"20.3.47-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 Mention module

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

```
`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Import the MentionModule for the Mention component from the dropdown package.
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
@NgModule({
  imports: [ BrowserModule, MentionModule ], // Registering EJ2 Mention Module.
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

Adding CSS reference

The following CSS files are available in `../node_modules/@syncfusion` package folder. Add Mentions component's styles as given below in `style.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/bootstrap5.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/bootstrap5.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/bootstrap5.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/bootstrap5.css';
@import '../node_modules/@syncfusion/ej2-angular-dropdowns/styles/bootstrap5.css';
`
```

Adding Mention component

Modify the template in the [src/app/app.component.ts] file to render the Mention component. Add the Angular Mention by using `<ejs-mention>` selector in `template` section of the app.component.ts file. To use the Mention component properly, the [target](#) property should be configured so that it renders the Mention component in the configured element.

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

@Component({
  selector: 'app-root',
  // Specifies the template string for the Mention component
  template: `
    <label style="font-size: 15px; font-weight: 600;">Comments</label>

    <!--Element which is the Mention component's target to list the suggestions-->

    <div id="mentionElement" style="min-height: 100px; border: 1px solid #D7D7D7; border-radius: 4px;
    padding: 8px; font-size: 14px; width: 600px;"></div>

    <ejs-mention [target]='mentionTarget'></ejs-mention>`
  })
  export class AppComponent {
    constructor() {
    }

    // Defines the target in which the Mention component is rendered.
    public mentionTarget: "#mentionElement";
  }
`
```

Binding data source

After initializing, populate the data in Mention using the [dataSource](#) property. Here, an array of string values is passed to the Mention component.

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

@Component({
  selector: 'app-root',
  // Specifies the template string for the Mention component
  template: `
    <label id="comment" >Comments</label>

    <!--Element which is the Mention component's target to list the suggestions-->
```

```

<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='userData' [target]='mentionTarget'></ejs-mention>`
})
export class AppComponent {
  constructor() {
  }
  // Defines the array of data.
  public userData: string[] = ['Selma Rose', 'Garth', 'Robert', 'William', 'Joseph'];
  // Defines the target in which the Mention component is rendered.
  public mentionTarget: string = "#mentionElement";
}
`

```

Running the application

Run the application in the browser using the following command:

```

`
ng serve
`

```

The following example shows a basic **Mention** component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  // Specifies the template string for the Mention component
  template: `
    <label id="comment" >Comments</label>
    <!--Element which is the Mention component's target to list the
    suggestions-->
    <div id="mentionElement" placeholder = "Type @ and tag user"></div>
    <ejs-mention [dataSource]='userData' [target]='mentionTarget'></ejs-
    mention>`,
})
export class AppComponent {
  constructor() {
  }
}

```

```
// Defines the array of data.
public userData: string[] = ['Selma Rose', 'Garth', 'Robert', 'William',
'Joseph'];
// Defines the target in which the mention component is rendered.
public mentionTarget:string = "#mentionElement";
}
```

MAIN.TS

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

Display Mention character

By using the [showMentionChar](#) property, the text content can be displayed along with the mention character. You can customize the mention character by using the [mentionChar](#) property in the Mention component.

By default, the [mentionChar](#) is @ and the [showMentionChar](#) property is disabled.

The following example displays the text content along with the mention character configured as #.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  // Specifies the template string for the Mention component
  template: `
<label id="comment" >Comments</label>
<!--Element which is the Mention component's target to list the
suggestions-->
<div id="mentionElement" placeholder = "Type @ and select your
comments"></div>
<ejs-mention [dataSource]='userData' [target]='mentionTarget'
[showMentionChar]="true" [mentionChar]="mentionCharacter"></ejs-mention>`,
})
export class AppComponent {
  constructor() {
  }
  // Defines the array of data.
  public userData: string[] = ['Selma Rose', 'Garth', 'Robert', 'William',
'Joseph'];
  // Defines the target in which the mention component is rendered.
  public mentionTarget: string = "#mentionElement";
  public mentionCharacter: string = "#";
}
```

```
}

```

MAIN.TS

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

Working with data in Angular Mention component

The Mention loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of either `array` or `DataManager`.

The Mention also supports different kinds of data services such as OData V4 and Web API, and data formats such as XML, JSON, and JSONP with the help of `DataManager` adaptors.

Fields	Type	Description
text	string	Specifies the display text of each list item.
value	number or string	Specifies the hidden data value mapped to each list item that should contain a unique value.
groupBy	string	Specifies the category under which the list item has to be grouped.
iconCss	string	Specifies the icon class of each list item.

When binding complex data to the Mention, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Binding local data

Local data can be represented in three ways as described in the following.

Array of simple data

The Mention has provided support to load an array of primitive data such as strings and numbers. Here, both the value and text fields act the same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='userData' [target]='mentionTarget' ></ejs-mention>`,
```



```

}))
export class AppComponent {
  constructor() {
  }
  // Defines the array of data.
  public userData: string[] = ['Selma Rose', 'Garth', 'Robert', 'William', 'Joseph'];
  // Defines the target in which the mention component is rendered.
  public mentionTarget: string = "#mentionElement";
}

```

MAIN.TS

```

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

```

Array of JSON data

The Mention can generate its list items through an array of complex data. Therefore, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **ID** column and **Game** column from complex data have been mapped to the **value** field and **text** field, respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { FieldSettingsModel } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag sport"></div>
<ejs-mention [dataSource]='sportsData' [target]='mentionTarget'
[fields]='fields' ></ejs-mention>`,
})
export class AppComponent {
  constructor() {
  }
  public sportsData: { [key: string]: Object }[] = [
    { ID: 'game1', Game: 'Badminton' },
    { ID: 'game2', Game: 'Football' },
    { ID: 'game3', Game: 'Tennis' },
    { ID: 'game4', Game: 'Hockey' },
    { ID: 'game5', Game: 'Basketball' }
  ];
  public mentionTarget: string = "#mentionElement";
}

```

```
public fields: Object = { text: 'Game', value: 'ID' }
}
```

MAIN.TS

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

Array of Complex data

The Mention can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, `Code.ID` column and `Country.Name` column from complex data have been mapped to the `value` field and `text` field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { FieldSettingsModel } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag country"></div>
<ejs-mention [dataSource]='countriesData' [target]='mentionTarget'
[fields]='fields' ></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public countriesData: { [key: string]: Object }[] = [
    { Country: { Name: 'Australia' }, Code: { ID: 'AU' } },
    { Country: { Name: 'Bermuda' }, Code: { ID: 'BM' } },
    { Country: { Name: 'Canada' }, Code: { ID: 'CA' } },
    { Country: { Name: 'Cameroon' }, Code: { ID: 'CM' } },
    { Country: { Name: 'Denmark' }, Code: { ID: 'DK' } },
    { Country: { Name: 'France' }, Code: { ID: 'FR' } },
  ];
  public mentionTarget: string = '#mentionElement';
  public fields: Object = { text: 'Country.Name', value: 'Code.ID' };
}
```

MAIN.TS

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

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

Binding remote data

The Mention supports retrieval of data from remote data services with the help of **DataManager** component. The [query](#) property is used to fetch the data from the database and bind it to the Mention component.

OData v4 adaptor - Binding OData v4 service

The ODataV4 is an improved version of OData protocols, and the **DataManager** can also retrieve and consume OData v4 services. For more details on OData v4 services, refer to the [odata documentation](#). To bind OData v4 service, use the **ODataV4Adaptor**.

The following sample displays the first 6 contacts from **Customers** table of the **Northwind** Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { FieldSettingsModel } from '@syncfusion/ej2-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='searchData' [query]='query' [fields]='fields'
[popupWidth]='popupWidth' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public searchData: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';
  public query: Query = new Query().from('Customers').select(['ContactName', 'CustomerID']).take(6);
  public fields: Object = { text: 'ContactName', value: 'CustomerID' };
  public popupWidth: string = '250px'
}
```

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 adaptor](#)

You can use **WebApiAdaptor** to bind mention with Web API created using OData endpoint.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='searchData' [query]='query' [fields]='fields'
[popupWidth]='popupWidth' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public searchData: DataManager = new DataManager({
    url: 'https://services.syncfusion.com/angular/production/api/Employees',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';
  public query: Query = new Query().select(['FirstName',
'EmployeeID']).take(7);
  public fields: Object = { text: 'FirstName', value: 'EmployeeID' };
  public popupWidth: string = '250px';
}
```

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

- [Customization](#)
- [How to perform filtering](#)

[Filtering data in Angular Mention component](#)

The Mention component has built-in support to filter data items. The filter operation starts as soon as you start typing characters in the mention element.

Limit the minimum filter character

You can control the minimum length of user input to initiate the search action using [minLength](#) property. This can be useful if you have a very large list of data. The default value is 0, where suggestion the list opened as soon as the user inputs the mention character.

The remote request does not fetch the search data until the search key contains three characters as shown in the following example.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='searchData' [query]='query' [fields]='fields'
[minLength]='minLen' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public searchData: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';
  public query: Query = new Query().select(['ContactName',
'CustomerID']).take(7);
  public fields: object = { text: 'ContactName', value: 'CustomerID' };
  public minLen: number = 3
}
```

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 filter type

While filtering, you can change the filter type to **Contains**, **StartsWith**, or **EndsWith** in the [filterType](#) property. The default filter operator is **Contains**.

- **StartsWith** - Filter the items that begin with the specified text value.
- **Contains** - Filter the items that contain the specified text value.

- **EndsWith** - Filter the items that end with the specified text value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>
  <ejs-mention [dataSource]='searchData' [query]='query' [fields]='fields'
  [filterType]='filterType' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public searchData: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';
  public query: Query = new Query().select(['ContactName',
  'CustomerID']).take(7);
  public fields: Object = { text: 'ContactName', value: 'CustomerID' };
  public filterType: string = 'EndsWith'
}
```

MAIN.TS

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

Allow spacing between search

While filtering the data in the data source, you can allow the space in the middle of the mention by using the [allowSpaces](#) property. If the data source does not match with the mentioned element data, the popup will be hidden on the space key press. The default value of `allowSpaces` is `false`.

By default, the `allowSpaces` property is disabled, and the space ends the mention component search.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
```

```
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>
  <ejs-mention [dataSource]='userData' [fields]='fields'
  [allowSpaces]='allowSpaces' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public userData: { [key: string]: Object }[] = [
    { Name : "Andrew Fuller", ID : "1" },
    { Name : "Anne Dodsworth" , ID : "2" },
    { Name : "Janet Leverling" , ID : "3" },
    { Name : "Laura Callahan" , ID : "4" },
    { Name : "Margaret Peacock" , ID : "5" }
  ];
  public fields: Object = {text:'Name'};
  public mentionTarget: string = '#mentionElement';
  public allowSpaces : 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));
```

Customize the suggestion item count

While filtering, you can customize the number of list items to be displayed in the suggestion list by using the [suggestionCount](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>
  <ejs-mention [dataSource]='emailData' [fields]='fields' [suggestionCount]=
  'suggestionCount' [target]='mentionTarget'></ejs-mention>`,
})
```

```
export class AppComponent {
  constructor() {}
  public emailData: { [key: string ] : Object }[] =[
    { Name: "Selma Rose", EmailId: "selma@gmail.com" },
    { Name: "Maria", EmailId: "maria@gmail.com" },
    { Name: "Russo Kay", EmailId: "russo@gmail.com" },
    { Name: "Robert", EmailId: "robert@gmail.com" },
    { Name: "Camden Kate",EmailId: "camden@gmail.com" },
    { Name: "Garth", EmailId: "garth@gmail.com" },
    { Name: "Andrew James", EmailId: "james@gmail.com" },
    { Name: "Olivia", EmailId: "olivia@gmail.com" },
    { Name: "Sophia", EmailId: "sophia@gmail.com" },
    { Name: "Margaret", EmailId: "margaret@gmail.com" },
    { Name: "Ursula Ann", EmailId: "ursula@gmail.com" },
    { Name: "Laura Grace", EmailId: "laura@gmail.com" },
    { Name: "Albert", EmailId: "albert@gmail.com" },
    { Name: "William", EmailId: "william@gmail.com" }
  ];
  public mentionTarget: string = '#mentionElement';
  public fields: Object = { text: 'Name' };
  public suggestionCount: Number = 8
}
```

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

- [Templates](#)

Sorting in Angular Mention component

You can display the suggestions list items in a specific order. It has possible types as **Ascending**, **Descending**, and **None** in the [sortOrder](#) property.

- **None** - The data source is not sorted.
- **Ascending** - The data source is sorted in ascending order.
- **Descending** - The data source is sorted in descending order.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
```



```

    ],
    standalone: true,
    selector: 'app-root',
    template: `<label id="comment" >Comments</label>
    <div id="mentionElement" placeholder = "Type @ and tag sport"></div>
    <ejs-mention [dataSource]='userData' [fields]='fields'
    [sortOrder]='sortOrder' [target]='mentionTarget'></ejs-mention>`,
  })
  export class AppComponent {
    constructor() {}
    public userData: { [key: string] : Object}[] = [
      { ID : "game1" ,Game : "Badminton"},
      { ID : "game2" ,Game : "Football" },
      { ID : "game3" ,Game : "Tennis"},
      { ID : "game4" ,Game : "Hockey"},
      { ID : "game5" ,Game : "Basketball"},
      { ID : "game6" ,Game : "Cricket"}
    ];
    public fields: Object = { text: 'Game' };
    public mentionTarget: string = '#mentionElement';
    public sortOrder: string = 'Descending';
  }

```

MAIN.TS

```

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

```

Template in Angular Mention component

The Mention has been provided with several options to customize each list item, display item and waiting popup. It uses the Essential JS 2 [Template engine](#) to compile and render the elements properly.

Item template

The content of each list item within the Mention can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>

```

```

<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='dataSource' [query]='query' [fields]='fields'
[target]='mentionTarget' [sortOrder]='sortOrder' popupWidth='250px'>
  <ng-template #itemTemplate let-data>
    <span><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></span></span>
  </ng-template></ejs-mention>`,
  })
export class AppComponent {
  constructor() {}
  public dataSource: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';
  public query: Query = new Query().from('Employees').select(['FirstName',
'City', 'EmployeeID']).take(6);
  public fields: object = { text: 'FirstName', value: 'EmployeeID' };
  public sortOrder: string = 'Ascending';
  public itemTemplate: any = "<span><span
class='name'>${FirstName}</span><span class ='city'>${City}</span></span>"
}

```

MAIN.TS

```

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

```

Display template

The currently selected value that is to be displayed on the mention element can be customized using the [displayTemplate](#) property.

In the following sample, the selected value is displayed as a combined text of both **FirstName** and **City** in the mention element, which is separated by a hyphen.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
`
})

```

```

    <ejs-mention [dataSource]='dataSource' [query]='query' [fields]='fields'
    [target]='mentionTarget' [sortOrder]='sortOrder' popupWidth='250px'>
      <ng-template #itemTemplate let-data>
        <span><span>{{data.FirstName}}</span><span class
        ='city'>{{data.City}}</span></span>
      </ng-template>
      <ng-template #displayTemplate let-data>
        <span>{{data.FirstName}} - {{data.City}}</span>
      </ng-template>
    </ejs-mention>,
  })
  export class AppComponent {
    constructor() {}
    public dataSource: DataManager = new DataManager({
      url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
      adaptor: new ODataV4Adaptor,
      crossDomain: true
    });
    public mentionTarget: string = '#mentionElement';
    public query: Query = new Query().from('Employees').select(['FirstName',
    'City', 'EmployeeID']).take(6);
    public fields: object = { text: 'FirstName', value: 'EmployeeID' };
    public sortOrder: string = 'Ascending';
    public itemTemplate: any = "<span><span>${FirstName}</span><span class
    ='city'>${City}</span></span>";
    public displayTemplate: any = "<span>${FirstName} - ${City}</span>"
  }

```

MAIN.TS

```

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

```

No records template

The Mention is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>

```

```

    <ejs-mention [dataSource]='userData' [target]='mentionTarget'
    [noRecordsTemplate]='noRecordsTemplate' ></ejs-mention>`,
  })
  export class AppComponent {
    constructor() {
    }
    public userData: string[] = [];
    public mentionTarget: string = "#mentionElement";
    public noRecordsTemplate: any = "<span class='norecord'> NO DATA
    AVAILABLE</span>"
  }

```

MAIN.TS

```

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

```

Spinner template

The Mention provides support to customize the waiting spinner when data fetching takes time to load respective data in the popup using the [spinnerTemplate](#) property.

In the following sample, customized spinner is shown while fetching the data from the service.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>
  <ejs-mention [dataSource]='searchData' [query]='query' [fields]='fields'
  [target]='mentionTarget' popupWidth='250px'>
    <ng-template #spinnerTemplate>
      <div class="spinner_loader"></div>
    </ng-template>
  </ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public searchData: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  public mentionTarget: string = '#mentionElement';

```

```

    public query: Query = new Query().from('Employees').select(['FirstName',
    'City', 'EmployeeID']).take(26);
    public fields: object = { text: 'FirstName', value: 'EmployeeID' };
    public sortOrder: string = 'Ascending';
    public spinnerTemplate:any = '<div class="spinner_loader"></div>'
  }

```

MAIN.TS

```

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

```

Localization in Angular Mention component

The Localization library allows you to localize static text content of the [noRecordsTemplate](#) properties according to the culture currently assigned to the Mention.

| Locale key | en-US (default) |

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

| noRecordsTemplate | No records found |

Loading translations

To load the translation object to your application, use the load function of the **L10n** class.

In the following sample, French culture is set to the mention component and no data is loaded. Hence, the [noRecordsTemplate](#) property displays its text in French culture initially.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
import { L10n } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
    <div id="mentionElement" placeholder = "Type @ and tag user"></div>
    <ejs-mention [dataSource]='customerData' [fields]='fields' [query]=
    'query' [locale]='locale' [target]='mentionTarget'></ejs-mention>`,
})
export class AppComponent implements OnInit {
  constructor() {}
  public customerData: DataManager = new DataManager({
    url:
    'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,

```

```

        crossDomain: true
    });
    public mentionTarget: string = '#mentionElement';
    public fields: Object = { text: 'ContactName', value: 'CustomerID' };
    public query: Query = new Query().select(['ContactName',
'CustomerID']).take(0);
    public locale: string = 'fr-BE';
    ngOnInit(): void {
        L10n.load({
            'fr-BE': {
                'mention': {
                    'noRecordsTemplate': "Aucun enregistrement trouvé",
                }
            }
        });
    }
}

```

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](#)
- [How to bind the data to the mention](#)

Customization in Angular Mention component

Show or hide mention character

You can show mention character as prefix of selected item in mention component using [showMentionChar](#) property. The default value of `ShowMentionChar` is `false`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag user"></div>
<ejs-mention [dataSource]='userData' [showMentionChar]='mentionShow'
[target]='mentionTarget' [fields]='fields' ></ejs-mention>`,
})

```

```
export class AppComponent {
  constructor() {}
  public userData: { [key: string]: Object }[] = [
    { Name : "Selma Rose", EmailId : "selma@gmail.com" },
    { Name : "Maria", EmailId : "maria@gmail.com" },
    { Name : "Russo kay", EmailId : "russo@gmail.com" },
    { Name : "Robert", EmailId : "robert@gmail.com" },
    { Name : "Garth", EmailId : "garth@gmail.com" }
  ];
  public fields: Object = {text:'Name'};
  public mentionTarget: string = '#mentionElement';
  public mentionShow: 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));
```

Adding the suffix character after selection

You can add suffix character while selecting an item in Mention component using [suffixText](#) property.

You can add space or new line as suffix to the selected item. The default values is empty string.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag country"></div>
  <ejs-mention [dataSource]='userData' [showMentionChar]='mentionShow'
  [suffixText]='textSuffix' [target]='mentionTarget' [fields]='fields' ></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public userData: { [key: string]: Object }[] = [
    { Country : "Australia", Code : "AU" },
    { Country : "Bermuda", Code : "BM" },
    { Country : "Canada", Code : "CA" },
    { Country : "Cameroon", Code : "CM" },
    { Country : "Denmark", Code : "DK" }
  ];
  public fields: Object = {text:'Country'};
  public mentionTarget: string = '#mentionElement';
```

```

    public mentionShow: boolean = true;
    public textSuffix: string = '&#160;';
}

```

MAIN.TS

```

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

```

Configure the popup list

You can customize the suggestion list width and height using the [popupHeight](#) and [popupWidth](#) properties.

By default, the popup list width value is set as `auto`. Depending on the mentioned suggestion data list, the width value is automatically adjusted. The popup list height value is set as `300px`.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns';
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
<div id="mentionElement" placeholder = "Type @ and tag sport"></div>
<ejs-mention [dataSource]='sportsData' [fields]='fields'
[target]='mentionTarget' [popupHeight]='height' [popupWidth]='width' ></ejs-mention>`,
})
export class AppComponent {
  constructor() {}
  public sportsData: {[key: string]: Object}[] = [
    { ID : "game1" ,Game : "Badminton"},
    { ID : "game2" ,Game : "Football" },
    { ID : "game3" ,Game : "Tennis"},
    { ID : "game4" ,Game : "Hockey"},
    { ID : "game5" ,Game : "Basketball"},
    { ID : "game6" ,Game : "Cricket"}
  ];
  public fields: Object = {text: 'Game'};
  public mentionTarget: string = '#mentionElement';
  public height: string = '100px';
  public width: string = '250px'
}

```

MAIN.TS


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

Trigger character

You can customize the trigger character by using the [mentionChar](#) property in the Mention component. The trigger character triggers the suggestion list to display in the target area.

By default, the [mentionChar](#) is @.

Accessibility in Angular Mention component

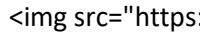
Web accessibility makes web content and web applications more accessible for people with disabilities. Mention component provides built-in compliance with WAI-ARIA specifications. The WAI-ARIA support is achieved using the attributes such as [aria-selected](#) and [aria-activedescendent](#).

The Mention 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 Mention component is outlined below.

| Accessibility Criteria | Compatibility |

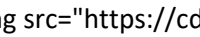
| -- | -- |

| [WCAG 2.2 Support](#) |  alt="Yes" > |

| [Section 508 Support](#) |  alt="Yes" > |

| [Screen Reader Support](#) |  alt="Yes" > |

| [Right-To-Left Support](#) |  alt="Yes" > |

| [Color Contrast](#) |  alt="Yes" > |

| [Mobile Device Support](#) |  alt="Yes" > |

| [Keyboard Navigation Support](#) |  alt="Yes" > |

| [Accessibility Checker Validation](#) |  alt="Yes" > |

| [Axe-core Accessibility Validation](#) |  alt="Yes" > |

<style>

```
.post .post-content img {
display: inline-block;
```

```
margin: 0.5em 0;
```

```
}
```

```
</style>
```

```
<div> - 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 Mention component uses the **Listbox** role where each list item has an **option** role. The following ARIA attributes denote the Mention state.

| Properties | Functionalities |

| --- | --- |

| aria-selected | Indicates the selected option. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

Keyboard interaction

You can use the following key shortcuts to access the Mention without interruptions.

| Keyboard shortcuts | Actions |

| --- | --- |

| Arrow Down | Selects the first item in the Mention list. Otherwise, selects the item next to the currently selected item. |

| Arrow Up | Selects the item previous to the currently selected one. |

| Esc(Escape) | Closes the popup list when it is in an open state. |

| Enter | Selects the focused item, and when it is in an open state the popup list closes. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, inserts the selected popup list item and closes the popup list. |

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MentionModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MentionModule
  ],
```

```

standalone: true,
  selector: 'app-root',
  template: `<label id="comment" >Comments</label>
  <div id="mentionElement" placeholder = "Type @ and tag user"></div>
  <ejs-mention [dataSource]='userData' [target]='mentionTarget'
  [fields]='fields'></ejs-mention>`,
))
export class AppComponent {
  constructor() {
  }
  public userData: { [key: string]: Object }[] = [
    { Name : "Andrew Fuller", ID : "1" },
    { Name : "Anne Dodsworth" , ID : "2" },
    { Name : "Janet Leverling" , ID : "3" },
    { Name : "Laura Callahan" , ID : "4" },
    { Name : "Margaret Peacock" , ID : "5" }
  ];
  public fields: Object = {text:'Name'};
  public mentionTarget: string = "#mentionElement";
}

```

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 Mention component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Mention component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Mention component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Menu Bar

Getting started with Angular Menu component

This section explains how to create a simple Menu, and demonstrate the basic usage of the Menu module in an Angular environment.

Dependencies

The list of dependencies required to use the Menu module in your application is given below:

```

`javascript
|-- @syncfusion/ej2-angular-navigations
|-- @syncfusion/ej2-angular-base

```

```
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-splitbuttons
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
\
```

Setup Angular environment

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

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

Create an Angular application

Start a new Angular application using below Angular CLI command.

```
\
ng new my-app
cd my-app
\
```

Installing Syncfusion Menu Package

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

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

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

Ivy library distribution package

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

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

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

Angular compatibility compiled package(ngcc)

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

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

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

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

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

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

Adding Menu module

Import Menu module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-navigations`.

```
`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Imported Syncfusion menu module from navigations package.
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { AppComponent } from './app.component';
@NgModule({
  imports: [BrowserModule, MenuModule], // Registering EJ2 Menu Module.
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule { }
`
```

Adding Syncfusion Menu component

Modify the template in `app.component.ts` file with `ejs-menu` to render the Menu component.

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

```
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  selector: 'app-root',
  template: `<!-- To Render Menu. -->
<ejs-menu [items]='menuItems'></ejs-menu>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' }
      ]
    }
  ],
}
```

```

{
  text: 'Tools',
  items: [
    { text: 'Spelling & Grammar' },
    { text: 'Customize' },
    { text: 'Options' }
  ],
  { text: 'Go' },
  { text: 'Help' }
];
}
`

```

Adding CSS reference

Add Menu component's styles as given below in `style.css`.

```

`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-navigations/styles/material.css';
`

```

Running the application

Run the application in the browser using the following command:

```

ng serve
`

```

The following example shows a basic `Menu` component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->

```

```

        <ejs-menu [items]='menuItems'></ejs-menu></div>`
    })
    export class AppComponent {
        public menuItems: MenuItemModel[] = [
            {
                text: 'File',
                items: [
                    { text: 'Open' },
                    { text: 'Save' },
                    { text: 'Exit' }
                ]
            },
            {
                text: 'Edit',
                items: [
                    { text: 'Cut' },
                    { text: 'Copy' },
                    { text: 'Paste' }
                ]
            },
            {
                text: 'View',
                items: [
                    { text: 'Toolbar' },
                    { text: 'Sidebar' }
                ]
            },
            {
                text: 'Tools',
                items: [
                    { text: 'Spelling & Grammar' },
                    { text: 'Customize' },
                    { text: 'Options' }
                ]
            },
            { text: 'Go' },
            { text: 'Help' }
        ];
    }

```

MAIN.TS

```

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

```

This example demonstrates the basic rendering of Menu with items support.

For more information about data source support, refer to the [Data Source Binding](#) section.

Group Menu items with separator

The separators are both horizontal and vertical lines used to separate the menu items.

You cannot select the separators, but you can enable separators to group the menu items using the [separator](#) property.

The **Open** and **Save** sub menu items are grouped using the **separator** property in the following sample.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems'></ejs-menu></div>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { separator: true },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' },
        { text: 'Full Screen' }
      ]
    },
    {
      text: 'Tools',
      items: [
        { text: 'Spelling & Grammar' },
        { text: 'Customize' },
        { text: 'Options' }
      ]
    }
  ],
}
```

```

        { text: 'Go' },
        { text: 'Help' }
    ];
}

```

MAIN.TS

```

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

```

The [separator](#) property should not be given along with the other fields in the [MenuItemModel](#).

You can also enable the separator to group **horizontal** menu items.

Icons and sub menu items in Angular Menu component

Icons

The menu item contains an icon/image in it to provide a visual representation of an action.

To place the icon on a menu item, set the [iconCss](#) property with the required icon CSS. By default, the icon is positioned at the left of the menu item. In the following sample, the icons of File and Edit menu items and Open, Save, Cut, Copy, and Paste sub menu items are added using the [iconCss](#) property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
    imports: [ MenuModule ],
    standalone: true,
    selector: 'app-root',
    template: `<div class="e-section-control">
        <ejs-menu #menu [items]='menuItems'></ejs-menu></div>`
})
export class AppComponent {
    //Menu items definition
    public menuItems: MenuItemModel[] = [
        {
            text: 'File',
            iconCss: 'em-icons e-file',
            items: [
                { text: 'Open', iconCss: 'em-icons e-open' },
                { text: 'Save', iconCss: 'e-icons e-save' },
                { separator: true },
                { text: 'Exit' }
            ]
        },
        {
            text: 'Edit',

```

```

        iconCss: 'em-icons e-edit',
        items: [
            { text: 'Cut', iconCss: 'em-icons e-cut' },
            { text: 'Copy', iconCss: 'em-icons e-copy' },
            { text: 'Paste', iconCss: 'em-icons e-paste' }
        ]
    },
    {
        text: 'View',
        items: [
            { text: 'Toolbar' },
            { text: 'Sidebar' },
            { text: 'Full Screen' }
        ]
    },
    {
        text: 'Tools',
        items: [
            { text: 'Spelling & Grammar' },
            { text: 'Customize' },
            { text: 'Options' }
        ]
    },
    { text: 'Go' },
    { text: 'Help' }
];
}

```

MAIN.TS

```

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

```

Navigation

Navigation in Menu is used to navigate to the other web page when a menu item is clicked.

It can be achieved by providing a link to the menu item using the [url](#) property. In the following sample, the Navigation URL is added to sub menu items using the url property.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">

```

```

        <ejs-menu #menu [items]='menuItems'
        (beforeItemRender)='beforeItemRender($event)'></ejs-menu></div>`
    })
    export class AppComponent {
        //Menu items definition
        public menuItems: MenuItemModel[] = [
            {
                text: 'Appliances',
                items: [
                    { text: 'Washing Machine', url:
'https://www.google.com/search?q=washing+machine' },
                    { text: 'Air Conditioners', url:
'https://www.google.com/search?q=air+conditioners' }
                ]
            },
            {
                text: 'Mobile',
                items: [
                    { text: 'Headphones', url:
'https://www.google.com/search?q=headphones' },
                    { text: 'Memory Cards', url:
'https://www.google.com/search?q=memory+cards' },
                    { text: 'Power Banks', url:
'https://www.google.com/search?q=power+banks' }
                ]
            },
            {
                text: 'Entertainment',
                items: [
                    { text: 'Televisions', url:
'https://www.google.com/search?q=televisions' },
                    { text: 'Home Theatres', url:
'https://www.google.com/search?q=home+theatres' },
                    { text: 'Gaming Laptops', url:
'https://www.google.com/search?q=gaming+laptops' }
                ]
            },
            { text: 'Fashion', url: 'https://www.google.com/search?q=fashion' },
            { text: 'Offers', url: 'https://www.google.com/search?q=offers' }
        ];
        public beforeItemRender(args: MenuEventArgs): void {
            if (args.item.url) {
                // To open url in blank page.
                args.element.getElementsByTagName('a')[0].setAttribute('target',
'_blank');
            }
        }
    }
}

```

MAIN.TS

```

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

```

Multilevel nesting

The Menu supports multiple level nesting, and it can be achieved by mapping the [items](#) property inside the parent [menuitems](#).

In the following sample, three-level nesting of menu has been provided.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu #menu [items]='menuItems'></ejs-menu></div>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'Fashion',
      items: [
        {
          text: 'Men Fashion',
          items: [
            {
              text: 'Personal Care',
              items: [
                { text: 'Trimmers' },
                { text: 'Shavers' }
              ]
            },
          ],
        },
        {
          text: 'Clothing',
          items: [
            { text: 'Shirts' },
            { text: 'Jackets' },
            { text: 'Track Suits' }
          ]
        },
        { text: 'Footwear' }
      ],
    },
    {
      text: 'Women Fashion',
      items: [
        {
          text: 'Clothing',
          items: [
            { text: 'Kurtas' },
            { text: 'Salwars' },
            { text: 'Sarees' }
          ]
        },
      ],
    },
  ],
}
```

```

    ],
    {
      text: 'Jewellery',
      items: [
        { text: 'Nosepins' },
        { text: 'Anklets' }
      ]
    }
  ],
  {
    text: 'Home & Living',
    items: [
      {
        text: 'Washing Machine',
        items: [
          { text: 'Fully Automatic' },
          { text: 'Semi Automatic' }
        ]
      },
      {
        text: 'Air Conditioners',
        items: [
          { text: 'Inverter ACs' },
          { text: 'Split ACs' }
        ]
      }
    ]
  },
  { text: 'Accessories' },
  { text: 'Sports' },
  { text: 'Gaming' }
];
}

```

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 achieve multi level nesting with data source by mapping **name** of the child items to the [children](#) sub-property of [fields](#) property. Also, we can specify [id](#) property for menu items. For more information, refer to the [data source binding](#) section.

The below table represents the MenuItem properties and its description.

Property Name	Type	Description
iconCss	string	Defines class/multiple classes separated by a space for the menu Item that is used to include an icon. Menu Item can include font icon and sprite image.

|id|string|Specifies the id for menu item.

|separator|boolean|Specifies separator between the menu items. Separator are either horizontal or vertical lines used to group menu items.

|items|MenuItemModel[]|Specifies the sub menu items that is the array of MenuItem model/

|text|string|Specifies text for menu item.

|url|string|Specifies url for menu item that creates the anchor link to navigate to the url provided.

See Also

- [Customize menu items](#)
- [Group menu items with separator](#)

Data source binding and custom menu items in Angular Menu component

Data binding

The Menu supports data source bindings such as array of JavaScript objects

that can be structured as either **hierarchical** or **self-referential** data.

Hierarchical data

The Menu can be populated with hierarchical data source by assigning it to the [items](#) property, and the fields with corresponding keys can be mapped to the [fields](#) property.

JSON data

The Menu can generate its menu items through an array of complex data source by mapping fields from the [fields](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { FieldSettingsModel } from '@syncfusion/ej2-angular-navigations';
// Import an array of JSON data from datasource.ts
import { dataSource } from './datasource';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu [items]="data" [fields]='menuFields'></ejs-menu></div>`
})
export class AppComponent {
  public menuFields: FieldSettingsModel = {
    text: ['continent', 'country', 'language'],
    children: ['countries', 'languages']
  };
  public data: { [key: string]: Object }[] = dataSource;
}
```

MAIN.TS

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

Data Service

In application level, remote data binding can be achieved using [DataManager](#).

To create Menu, assign items property with resultant data from [callback](#) function.

The following example displays five employees' **FirstName** from **Employees** table and **ShipName** details from **Orders** table of the **Northwind** Data Service.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component, OnInit } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { DataManager, Query, ODataV4Adaptor, ReturnOption } from
 '@syncfusion/ej2-data';
import { FieldSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu *ngIf='menuItems' [items]='menuItems'
    [fields]='menuFields'></ejs-menu></div>`
})
export class AppComponent implements OnInit {
  private SERVICE_URI: string =
    'https://services.odata.org/V4/Northwind/Northwind.svc/';
  // Menu fields definition.
  public menuFields: FieldSettingsModel = {
    text: ['FirstName', 'ShipName'],
    children: ['Orders']
  };
  public menuItems?: { [key: string]: Object }[];
  public ngOnInit(): void {
    // Getting remote data using DataManager.
    new DataManager({ url: this.SERVICE_URI, adaptor: new
    ODataV4Adaptor(), crossDomain: true })
    .executeQuery(
      new Query().from('Employees').take(5).hierarchy(
        new Query()
          .foreignKey('EmployeeID')
          .from('Orders').take(13),
        function () {
          return [1, 2, 3, 4, 5]
        }
      )
    )
    .then((e: ReturnOption) => {
```



```

        //Assign result data to menu items
        this.menuItems = e.result as { [key: string]: Object }[];
        const loaderElement = document.getElementById('loader') as
HTMLInputElement;
        if (loaderElement) {
            loaderElement.style.display = "none";
        }
    });
}
}

```

MAIN.TS

```

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

```

Self-referential data

Menu can be populated from self-referential data structure that contains array of JSON objects with `parentId` mapping.

You can directly assign self-referential data to the `items` property, and map all the field members with corresponding keys from self-referential data to `fields` property.

To render the root level nodes, specify the `parentId` as null or no need to specify the `parentId` in data source.

In the following example, the `id`, `pId`, and `text` columns from self-referential data have been mapped to the `itemId`, `parentId`, and `text` fields, respectively.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { FieldSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu [items]='data' [fields]='menuFields'></ejs-menu>
  </div>`
})
export class AppComponent {
  //Menu datasource
  public data: { [key: string]: Object }[] = [
    { id: 'parent1', text: 'Events' },
    { id: 'parent2', text: 'Movies' },
    { id: 'parent3', text: 'Directory' },
    { id: 'parent4', text: 'Queries', pId: null as any },
    { id: 'parent5', text: 'Services', pId: null as any },
  ]
}

```

```

    { id: 'parent6', text: 'Conferences', pId: 'parent1' },
    { id: 'parent7', text: 'Music', pId: 'parent1' },
    { id: 'parent8', text: 'Workshops', pId: 'parent1' },
    { id: 'parent9', text: 'Now Showing', pId: 'parent2' },
    { id: 'parent10', text: 'Coming Soon', pId: 'parent2' },
    { id: 'parent10', text: 'Media Gallery', pId: 'parent3' },
    { id: 'parent11', text: 'Newsletters', pId: 'parent3' },
    { id: 'parent12', text: 'Our Policy', pId: 'parent4' },
    { id: 'parent13', text: 'Site Map', pId: 'parent4' },
    { id: 'parent14', text: 'Pop', pId: 'parent7' },
    { id: 'parent15', text: 'Folk', pId: 'parent7' },
    { id: 'parent16', text: 'Classical', pId: 'parent7' }
  ];
  //Menu fields definition
  public menuFields: FieldSettingsModel = {
    itemId: 'id',
    text: 'text',
    parentId: 'pId'
  };
}

```

MAIN.TS

```

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

```

Custom menu items

The Menu can be customized using Essential JS2 [Template engine](#) to render the elements.

To customize menu items in your application, set your customized template string to the [template](#) property.

In the following example, the menu has been rendered with customized menu items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject } from '@angular/core';
import { FieldSettingsModel } from '@syncfusion/ej2-angular-navigations';
import { enableRipple } from '@syncfusion/ej2-base';
enableRipple(false);
@Component({
  imports: [ MenuModule, ButtonModule],
  providers: [
    { provide: 'sourceFiles', useValue: {files: []}
  ],
  standalone: true,
  selector: 'app-root',
  styleUrls: ['./template.css'],
  template: `<div class="e-section-control">
<div id="menuTemplate" class="menu-section">
<div class="menu-control">

```

```

    <ejs-menu [items]='dataSource' [fields]='menuFields'
[animationSettings]="animation" cssClass="e-template-menu">
    <ng-template #template let-datasource="">
        {{datasource.category}}
        <div *ngIf="datasource.value"
style="width:100%;display:flex;justify-content:space-between;">
            
            <span style="width:100%;">{{datasource.value}}</span>
            <span *ngIf="datasource.count" class='e-badge e-badge-
success'>{{datasource.count}}</span>
        </div>
        <div *ngIf="datasource.about" tabindex="0" class="e-card">
            <div class="e-card-header">
                <div class="e-card-header-caption">
                    <div class="e-card-header-title">About Us</div>
                </div>
            </div>
            <div class="e-card-content">
                {{datasource.about.value}}
            </div>
            <div class="e-card-actions">
                <button class="e-btn e-outline" style="pointer-
events: auto;">
                    Read More
                </button>
            </div>
        </div>
    </ng-template>
</ejs-menu>
</div>
</div>`
}))
export class AppComponent {
animation: any;
constructor(@Inject('sourceFiles') public sourceFiles: any) {
    sourceFiles.files = ['template.css'];
}
//Template datasource
public dataSource: { [key: string]: Object }[] = [
    {
        category: 'Products',
        options: [
            { value: 'JavaScript', url:
'https://ej2.syncfusion.com/angular/documentation/../../../../menu/images/
javascript' },
            { value: 'Angular', url:
'../../../../menu/images/angular' },
            { value: 'ASP.NET Core', url:
'../../../../menu/images/core' },
            { value: 'ASP.NET MVC', url:
'../../../../menu/images/mvc' }
        ],
    },
    {

```

```

        category: 'Services',
        options: [
            { value: 'Application Development', count: '1200+' },
            { value: 'Maintenance & Support', count: '3700+' },
            { value: 'Quality Assurance' },
            { value: 'Cloud Integration', count: '900+' }
        ]
    },
    {
        category: 'About Us',
        options: [
            {
                id: 'about',
                about: {
                    value: "We are on a mission to provide world-class
best software solutions for web, mobile and desktop platforms. Around 900+
applications are desgined and delivered to our customers to make digital &
strengthen their businesses."
                }
            }
        ]
    },
    { category: 'Careers' },
    { category: 'Sign In' }
];
// Menu fields definition
public menuFields: object = {
    text: ['category', 'value'],
    children: ['options']
};
}

```

MAIN.TS

```

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

```

To prevent sub menu closing, set `args.cancel` to `true` in [beforeClose](#) event.

See Also

- [Render menu with items](#)

Use case scenarios in Angular Menu component

Scrollable menu

The menu component supports horizontal and vertical scrolling to render large menus and submenus in an adaptive way. This can be achieved by enabling the [enableScrolling](#) property and by restricting the corresponding menu/submenu size.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```

import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple, closest } from '@syncfusion/ej2-base';
import { MenuItemModel, BeforeOpenCloseMenuEventArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu [items]='menuItems' cssClass='e-scrollable-menu'
[enableScrolling]="true" (beforeOpen)='onBeforeOpen($event)'></ejs-menu>
  </div>`
})
export class AppComponent {
  menuItems: MenuItemModel[] = [
    {
      text: 'Appliances',
      items: [
        {
          text: 'Kitchen',
          items: [
            { text: 'Electric Cookers' },
            { text: 'Coffee Makers' },
            { text: 'Blenders' },
            { text: 'Microwave Ovens' }
          ]
        },
        {
          text: 'Television',
          items: [
            { text: 'Our Exclusive TVs' },
            { text: 'Smart TVs' },
            { text: 'Big Screen TVs' }
          ]
        },
        {
          text: 'Washing Machine'
        },
        {
          text: 'Refrigerators'
        },
        {
          text: 'Air Conditioners',
          items: [
            { text: 'Inverter ACs' },
            { text: 'Split ACs' },
            { text: 'Window ACs' },
          ]
        },
        {
          text: 'Water Purifiers'
        },
        {
          text: 'Air Purifiers'
        }
      ]
    }
  ]
}

```

```

    },
    {
      text: 'Chimneys'
    },
    {
      text: 'Inverters'
    },
    {
      text: 'Healthy Living'
    },
    {
      text: 'Vacuum Cleaners'
    },
    {
      text: 'Room Heaters'
    },
    {
      text: 'New Launches'
    }
  ]
},
{
  text: 'Accessories',
  items: [
    {
      text: 'Mobile',
      items: [
        { text: 'Headphones' },
        { text: 'Memory Cards' },
        { text: 'Power Banks' },
        { text: 'Mobile Cases' },
        { text: 'Screen Protectors' }
      ]
    },
    {
      text: 'Laptops'
    },
    {
      text: 'Desktop PC',
      items: [
        { text: 'Pendrives' },
        { text: 'External Hard Disks' },
        { text: 'Monitors' },
        { text: 'Keyboards' }
      ]
    },
    {
      text: 'Camera',
      items: [
        { text: 'Lens' },
        { text: 'Tripods' }
      ]
    }
  ]
},
{
  text: 'Fashion',

```

```

        items: [
            {
                text: 'Men'
            },
            {
                text: 'Women'
            }
        ]
    },
    {
        text: 'Home & Living',
        items: [
            {
                text: 'Furniture'
            },
            {
                text: 'Decor'
            },
            {
                text: 'Smart Home Automation'
            },
            {
                text: 'Dining & Serving'
            }
        ]
    },
    {
        text: 'Entertainment',
        items: [
            {
                text: 'Televisions'
            },
            {
                text: 'Home Theatres'
            },
            {
                text: 'Gaming Laptops'
            }
        ]
    },
    {
        text: 'Contact Us'
    },
    {
        text: 'Help'
    }
];
onBeforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
    // Restricting sub menu wrapper height
    if (args.parentItem.text === 'Appliances') {
        (closest(args.element, '.e-menu-wrapper') as
HTMLInputElement).style.height = '230px';
    }
}
}

```

MAIN.TS

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

Menu in toolbar

The following example demonstrates how to integrate Menu with Toolbar component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule, ToolbarModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { ToolbarComponent, MenuAnimationSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule, ToolbarModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="control-section">
      <div class="toolbar-menu-control">
        <div id='menu'><ejs-menu [items]='menuItems'
[fields]='menuFields' [animationSettings]='animationSettings'></ejs-menu></div>
        <ejs-toolbar id="shoppingtoolbar" #toolbar
(created)='created()'>
          <e-items>
            <e-item template='#menu'></e-item>
            <e-item prefixIcon='em-icons e-shopping-cart'
align='Right'></e-item>
          </e-items>
        </ejs-toolbar>
      </div>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('toolbar')
  public toolbarObj?: ToolbarComponent;
  public menuItems: { [key: string]: Object }[] = [
    {
      header: 'Events',
      subItems: [
        { text: 'Conferences' },
        { text: 'Music' },
        { text: 'Workshops' }
      ]
    },
  ],
}
```



```

        header: 'Movies',
        subItems: [
            { text: 'Now Showing' },
            { text: 'Coming Soon' }
        ]
    },
    {
        header: 'Directory',
        subItems: [
            { text: 'Media Gallery' },
            { text: 'Newsletters' }
        ]
    },
    {
        header: 'Queries',
        subItems: [
            { text: 'Our Policy' },
            { text: 'Site Map' },
            { text: '24x7 Support' }
        ]
    },
    { header: 'Services' }
];
public menuFields: Object = {
    text: ['header', 'text', 'value'],
    children: ['subItems', 'options']
};
public animationSettings: MenuAnimationSettingsModel = { effect: 'None'
};
public created(): void {
    this.toolbarObj?.refreshOverflow();
}
}

```

MAIN.TS

```

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

```

Hamburger menu

The following example demonstrates the use case of menu with Accordion component integrated in SideBar.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { SidebarModule, AccordionModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { Accordion } from '@syncfusion/ej2-navigations';

```

```

import { SidebarComponent, AccordionComponent, ExpandEventArgs,
AccordionClickArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
imports: [ SidebarModule, AccordionModule],
standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="header">
      <span id="hamburger" class="e-icons menu default"
(click)="hamburgerClick()"></span>
      <div class="content">Header content</div>
    </div>
    <ejs-sidebar #sidebar id='default'-sidebar' width='220px' type='Over'>
      <div class="title-header">
        <div style="display:inline-block"> Menu </div>
        <span id="close" class="e-icons" (click)="close()"></span>
      </div>
      <div class="content-area">
        <ejs-accordion #accordion [items]='data'
(expanding)='expand($event)' (clicked)='clicked($event)'></ejs-accordion>
      </div>
    </ejs-sidebar>
    <!-- main content declaration -->
    <div>
      <div class="main-content">Main content</div>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('sidebar')
  public sidebarObj?: SidebarComponent;
  @ViewChild('accordion')
  public accordionObj?: AccordionComponent;
  public data: { [key: string]: Object }[] = [
    {
      header: 'Appliances',
      content: '<div id="Appliances_Items"></div>',
      subItems: [
        {
          header: 'Kitchen',
          content: '<div id="Appliances_Kitchen_Items"></div>',
          subItems: [
            { header: 'Electric Cookers' },
            { header: 'Coffee Makers' },
            { header: 'Blenders' },
          ]
        },
        {
          header: 'Washing Machine',
          content: '<div id="Appliances_Washing_Items"></div>',
          subItems: [
            { header: 'Fully Automatic' },
            { header: 'Semi Automatic' }
          ]
        }
      ]
    }
  ]
}

```

```

        header: 'Air Conditioners',
        content: '<div
id="Appliances_Conditioners_Items"></div>',
        subItems: [
            { header: 'Inverter ACs' },
            { header: 'Split ACs' },
            { header: 'Window ACs' },
        ]
    },
],
{
    header: 'Accessories',
    content: '<div id="Accessories_Items"></div>',
    subItems: [
        {
            header: 'Mobile',
            content: '<div id="Accessories_Mobile_Items"></div>',
            subItems: [
                { header: 'Headphones' },
                { header: 'Memory Cards' },
                { header: 'Power Banks' }
            ]
        },
        {
            header: 'Computer',
            content: '<div id="Accessories_Computer_Items"></div>',
            subItems: [
                { header: 'Pendrives' },
                { header: 'External Hard Disks' },
                { header: 'Monitors' }
            ]
        }
    ]
},
{
    header: 'Fashion',
    content: '<div id="Fashion_Items"></div>',
    subItems: [
        { header: 'Men' },
        { header: 'Women' }
    ]
},
{
    header: 'Home & Living',
    content: '<div id="Home_Living_Items"></div>',
    subItems: [
        { header: 'Furniture' },
        { header: 'Decor' }
    ]
},
{
    header: 'Entertainment',
    content: '<div id="Entertainment_Items"></div>',
    subItems: [
        { header: 'Televisions' },
        { header: 'Home Theatres' },
    ]
}

```

```

        { header: 'Gaming Laptops' }
    ]
}
];
//Expanding Event function for Accordion component.
public expand(e: ExpandEventArgs): void {
    if (e.isExpanded) {
        if ((e.element as HTMLElement).getElementsByClassName('e-acrdn-content')[0].children[0].classList.contains('e-accordion')) {
            return;
        }
        //Initialize Nested Accordion component
        let nestAcrdn: Accordion = new Accordion({
            items: (<{ subItems: object[] }>e.item).subItems,
            expanding: this.expand,
            clicked: this.clicked
        });
        let elemId: string = (e.element as
        HTMLElement).getElementsByClassName('e-acrdn-content')[0].children[0].id;
        //Render initialized Nested Accordion component
        nestAcrdn.appendTo('#' + elemId);
    }
}
public clicked(e: AccordionClickArgs): void {
    if (!e.item && !((e.originalEvent as Event).target as
    HTMLElement).closest('.e-acrdn-item') as Element).getElementsByClassName('e-
    tgl-collapse-icon').length) {
        this.sidebarObj?.hide();
    }
}
public hamburgerClick(): void {
    this.sidebarObj?.show();
    (this.accordionObj as AccordionComponent).refresh();
}
public close(): void {
    this.sidebarObj?.hide();
}
}

```

MAIN.TS

```

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

```

Mobile view

The following example demonstrates the use case of Menu in Mobile mode by using ListView component with hamburger.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { ListViewModule } from '@syncfusion/ej2-angular-lists'

```

```

import { Component, ViewChild } from '@angular/core';
import { enableRipple, Animation, AnimationOptions } from '@syncfusion/ej2-base';
import { ListViewComponent, SelectEventArgs } from '@syncfusion/ej2-angular-lists';
enableRipple(true);
@Component({
  imports: [ ListViewModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="layoutWrapper">
      <div class="speaker">
        <div class="camera"></div>
      </div>
      <div class="layout">
        <div id="container">
          <div id="header">
            <span id="hamburger" (click)="hamburgerClick()"
class="e-icons menu default"></span>
            <div class="content">Header</div>
          </div>
          <!-- ListView element -->
          <ejs-listview id="listview" #listview
[dataSource]="dataSource" headerTitle="Menu" [showHeader]="true"
(select)="onSelect($event)" tabindex="1" [style.display]='listViewDisplay'
></ejs-listview>
          <span id="close" (click)="onClick($event)" class="e-
icons" [style.display]='closeSpanDisplay'></span>
        </div>
      </div>
      <div class="outerButton"> </div>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('listview')
  public listObj?: ListViewComponent;
  public listViewDisplay: string = 'none';
  public closeSpanDisplay: string = 'none';
  public dataSource: { [key: string]: Object }[] = [
    {
      text: 'Appliances',
      id: 'list1',
      child: [
        {
          text: 'Kitchen',
          id: 'list1_1',
          child: [
            { id: 'list1_1_1', text: 'Electric Cookers' },
            { id: 'list1_1_2', text: 'Coffee Makers' },
            { id: 'list1_1_3', text: 'Blenders' },
          ]
        },
      ],
    },
    {
      text: 'Washing Machine',
      id: 'list1_2',
    }
  ]
}

```

```

        child: [
          { id: 'list1_2_1', text: 'Fully Automatic' },
          { id: 'list1_2_2', text: 'Semi Automatic' }
        ]
      },
      {
        text: 'Air Conditioners',
        id: 'list1_3',
        child: [
          { id: 'list1_3_1', text: 'Inverter ACs' },
          { id: 'list1_3_2', text: 'Split ACs' },
          { id: 'list1_3_3', text: 'Window ACs' },
        ]
      }
    ]
  },
  {
    text: 'Accessories',
    id: 'list2',
    child: [
      {
        text: 'Mobile',
        id: 'list2_1',
        child: [
          { id: 'list2_1_1', text: 'Headphones' },
          { id: 'list2_1_2', text: 'Memory Cards' },
          { id: 'list2_1_3', text: 'Power Banks' }
        ]
      },
      {
        text: 'Computer',
        id: 'list2_2',
        child: [
          { id: 'list2_2_1', text: 'Pendrives' },
          { id: 'list2_2_2', text: 'External Hard Disks' },
          { id: 'list2_2_3', text: 'Monitors' }
        ]
      }
    ]
  },
  {
    text: 'Fashion',
    id: 'list3',
    child: [
      { id: 'list3_1', text: 'Men' },
      { id: 'list3_2', text: 'Women' }
    ]
  },
  {
    text: 'Home & Living',
    id: 'list4',
    child: [
      { id: 'list4_1', text: 'Furniture' },
      { id: 'list4_2', text: 'Decor' }
    ]
  },
  {

```

```

        text: 'Entertainment',
        id: 'list5',
        child: [
            { id: 'list5_1', text: 'Televisions' },
            { id: 'list5_2', text: 'Home Theatres' },
            { id: 'list5_3', text: 'Gaming Laptops' }
        ]
    };
    public onSelect(e: SelectEventArgs): void {
        if (e.data && !(e.data as { child: object }).child) {
            this.listViewDisplay = 'none';
            this.closeSpanDisplay = 'none';
            this.listObj?.refresh();
        }
    }
    public onClick = (args: any): void => {
        this.listViewDisplay = 'none';
        this.closeSpanDisplay = 'none';
    };
    public hamburgerClick = () => {
        let animation: Animation = new Animation({ duration: 500 });
        animation.animate((this.listObj as ListViewComponent).element, {
            name: 'SlideDown',
            begin: (args: AnimationOptions) => {
                this.listViewDisplay = 'block';
                this.closeSpanDisplay = 'block';
            }
        });
    };
}

```

MAIN.TS

```

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

```

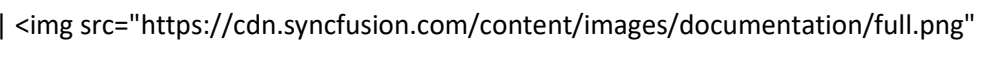
Accessibility in Angular Menu component

The Menu component followed the accessibility guidelines and standards, including [ADA, Section 508](#), [WCAG 2.2](#) standards, and [WCAG roles](#) that are commonly used to evaluate accessibility.

The accessibility compliance for the Menu component is outlined below.

| Accessibility Criteria | Compatibility |

| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes"> |

| [Section 508](#) Support |  alt="Yes"> |

```

| 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 Menu component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Menu component:

- | Attributes | Purpose |
- | --- | --- |
- | `role` | Indicates Menu component's root menu as `menubar`, popup as `menu`, and the popup items as `menuitem`. |
- | `aria-haspopup` | Indicates the availability and type of interactive popup element. |
- | `aria-expanded` | Indicates whether the subtree can be expanded or collapsed, and indicates whether its current state can be expanded or collapsed. |

| **aria-orientation** | Indicates whether the orientation is horizontal or vertical. The default orientation is horizontal. |

| **aria-label** | Indicates the menu item text. |

| **aria-disabled** | Indicates the state of menu item whether it is disabled. |

Keyboard interaction

The Menu component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Menu component.

| **Press** | **To do this** |

| --- | --- |

| **Esc** | Closes the sub menu that contains focus and returns focus to the parent element. |

| **Enter** | Opens the sub menu if focused menu item has sub menu, and places focus on its first item or activates the item and closes the sub menu. |

| **Up** | Navigates up or to the previous menu item. |

| **Down** | Navigates down or to the next menu item. |

| **Left** | Closes the current sub menu and navigates to the parent menu. |

| **Right** | Navigates and open the next sub menu. |

| **Home** | Focuses the first item. |

| **End** | Focuses the last item. |

Ensuring accessibility

The Menu component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Menu component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Menu component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Style and appearance in Angular Menu component

To modify the Menu appearance, you need to override the default CSS of Menu component. Please find the list of CSS classes and its corresponding section in Menu component. Also, you have an option to create your own custom theme for the controls using our [Theme Studio](#).

CSS Class | Purpose of Class

|.e-menu-wrapper|To customize the menu wrapper

|.e-menu-wrapper.e-menu-popup|To customize the menu popup wrapper

|.e-menu-wrapper ul .e-menu-item|To customize the menu items

|.e-menu-wrapper.e-menu-popup .e-menu-item|To customize the menu popup items

|.e-menu-wrapper ul .e-menu-item .e-caret|To customize the menu items caret icon

How To

Change orientation in Angular Menu component

Orientation in menu items can be changed horizontally or vertically using the [orientation](#) property.

By default, it is horizontally aligned.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems' orientation='Vertical'></ejs-
menu></div>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' },
        { text: 'Full Screen' }
      ]
    },
    {
      text: 'Tools',
      items: [
        { text: 'Spelling & Grammar' },
        { text: 'Customize' },
```

```

        { text: 'Options' }
    ],
    { text: 'Help' }
];
}

```

MAIN.TS

```

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

```

Customize menu using events in Angular Menu component

The Menu provides a set of [events](#) to enable users to customize it.

The available events are [beforeOpen](#), [beforeClose](#), [onClose](#), [onOpen](#), and [select](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel, OpenCloseMenuEventArgs, BeforeOpenCloseMenuEventArgs,
MenuEventArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="control-section">
      <div class="menu-section">
        <ejs-menu id='menu' [items]='menuItems'
(beforeOpen)='beforeOpen($event)' (beforeClose)='beforeClose($event)'
(onClose)='onClose($event)' (onOpen)='onOpen($event)'
(select)='select($event)'></ejs-menu>
      </div>
      <div class="property-section">
        <table id="propertyTable" title="Event trace">
          <tbody>
            <th>Event trace:</th>
            <tr>
              <td [innerHTML]="eventTrace"></td>
            </tr>
          </tbody>
        </table>
      </div>
      <button id='clear' ej2-button cssClass='e-small'
(click)='btnClick()'>Clear</button>
    </div>
  `
})

```

```

    })
    export class AppComponent {
        public eventTrace: string = '';
        public menuItems: MenuItemModel[] = [
            {
                text: 'Events',
                items: [
                    { text: 'Conferences' },
                    { text: 'Music' },
                    { text: 'Workshops' }
                ]
            },
            {
                text: 'Movies',
                items: [
                    { text: 'Now Showing' },
                    { text: 'Coming Soon' }
                ]
            },
            {
                text: 'Directory',
                items: [
                    { text: 'Media Gallery' },
                    { text: 'Newsletters' }
                ]
            },
            {
                text: 'Queries',
                items: [
                    { text: 'Our Policy' },
                    { text: 'Site Map' }
                ]
            },
            { text: 'Services' }
        ];
        public beforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
            this.updateEventLog(args);
        }
        public beforeClose(args: BeforeOpenCloseMenuEventArgs): void {
            this.updateEventLog(args);
        }
        public onClose(args: OpenCloseMenuEventArgs): void {
            this.updateEventLog(args);
        }
        public onOpen(args: OpenCloseMenuEventArgs): void {
            this.updateEventLog(args);
        }
        public select(args: MenuEventArgs): void {
            this.updateEventLog(args);
        }
        public updateEventLog(args: any): void {
            this.eventTrace = this.eventTrace + args.name + ' Event triggered.
<br />'
        }
        public btnClick(): void {
            this.eventTrace = '';
        }
    }

```

```
}

```

MAIN.TS

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

Customize menu items in Angular Menu component

Add or remove menu items

Menu items can be added or removed using the [insertAfter](#), [insertBefore](#) and [removeItems](#) methods.

In the following example, the **Europe** menu items are added before the **Oceania** item, the **Africa** menu items are added after the **Asia**, and the **South America** and **Mexico** items are removed from menu.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component, ViewChild } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuComponent, FieldSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu #menu [items]='data' [fields]='menuFields'
(created)='created()'></ejs-menu></div>`
})
export class AppComponent {
  @ViewChild('menu')
  public menuObj?: MenuComponent;
  //Menu datasource
  public data: { [key: string]: Object }[] = [
    {
      continent: 'Asia',
      countries: [
        { country: 'China' },
        { country: 'India' },
        { country: 'Japan' }
      ]
    },
    {
      continent: 'North America',
      countries: [
        { country: 'Canada' },
        { country: 'Mexico' },
        { country: 'USA' }
      ]
    }
  ]
}
```

```

    },
    {
        continent: 'South America',
        countries: [
            { country: 'Brazil' },
            { country: 'Colombia' },
            { country: 'Argentina' }
        ]
    },
    {
        continent: 'Oceania',
        countries: [
            { country: 'Australia' },
            { country: 'New Zealand' },
            { country: 'Samoa' },
        ]
    },
    { continent: 'Antarctica' }
];
//Menu fields definition
public menuFields: FieldSettingsModel = {
    text: ['continent', 'country'],
    children: ['countries']
};
public created(): void {
    let insertItem: { [key: string]: Object }[] = [
        {
            continent: 'Europe',
            countries: [
                { country: 'Finland' },
                { country: 'Austria' }
            ]
        }
    ];
    //Add items before to 'Oceania'
    this.menuObj?.insertBefore(insertItem, 'Oceania', false);
    insertItem = [
        {
            continent: 'Africa',
            countries: [
                { country: 'Nigeria' }
            ]
        }
    ];
    //Add items after to 'Asia'
    this.menuObj?.insertAfter(insertItem, 'Asia', false);
    //Remove items
    this.menuObj?.removeItems(['South America', 'Mexico'], false);
}
}

```

MAIN.TS

```

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

```

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

To process items with `ID` values, set `isUnique` to `true`.

Enable or disable menu items

You can enable and disable the menu items using the [enableItems](#) method in Menu. To enable menuItems, set the `enable` property in argument to `true` and vice-versa.

In the following example, the **Directory** header item, **Conferences**, and **Music** sub menu items are disabled.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuComponent, MenuItemModel, BeforeOpenCloseMenuEventArgs } from
 '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="control-section">
      <button ej2-button (click)="btnClick()">Enable all items</button>
      <div class="menu-section">
        <ejs-menu #menu [items]='menuItems'
  (beforeOpen)="beforeOpen($event)" (created)="created()"></ejs-menu>
      </div>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('menu')
  public menuObj?: MenuComponent;
  //Menu items definition
  public menuItems: MenuItemModel[] = [
    {
      text: 'Events',
      items: [
        { text: 'Conferences' },
        { text: 'Music' },
        { text: 'Workshops' }
      ]
    },
    {
      text: 'Movies',
      items: [
        { text: 'Now Showing' },
        { text: 'Coming Soon' }
      ]
    }
  ],
```

```

        {
            text: 'Directory',
            items: [
                { text: 'Media Gallery' },
                { text: 'Newsletters' }
            ]
        },
        {
            text: 'Queries',
            items: [
                { text: 'Our Policy' },
                { text: 'Site Map' }
            ]
        },
        { text: 'Services' }
    ];
    public disableItems: string[] = ['Conferences', 'Music', 'Directory'];
    public beforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
        //Handling sub menu items
        for (let i: number = 0; i < args.items.length; i++) {
            if (this.disableItems.indexOf(args.items[i].text as string) > -1)
            {
                this.menuObj?.enableItems([args.items[i].text as string],
                false, false);
            }
        }
        public created(): void {
            //Disable items
            this.menuObj?.enableItems(this.disableItems, false, false);
        }
        public btnClick(): void {
            //Enable items
            this.menuObj?.enableItems(this.disableItems, true, false);
            this.disableItems = [];
        }
    }
}

```

MAIN.TS

```

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

```

To disable sub menu items, use the [beforeOpen](#) event.

Hide or show menu items

You can show or hide the menu items using the [showItems](#) and [hideItems](#) methods.

In the following example, the **Movies** header item, **Workshops**, and **Music** sub menu items are hidden in menu.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```



```

import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuComponent, MenuItemModel, BeforeOpenCloseMenuEventArgs } from
 '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule, ButtonModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="control-section">
      <button ej2-button (click)= 'btnClick()' >Show all items</button>
      <div class="menu-section">
        <ejs-menu #menu [items]='menuItems'
(beforeOpen)= 'beforeOpen($event)' (created)= 'created()' ></ejs-menu>
      </div>
    </div>
  </div>`
})
export class AppComponent {
  @ViewChild('menu')
  public menuObj?: MenuComponent;
  public menuItems: MenuItemModel[] = [
    {
      text: 'Events',
      items: [
        { text: 'Conferences' },
        { text: 'Music' },
        { text: 'Workshops' }
      ]
    },
    {
      text: 'Movies',
      items: [
        { text: 'Now Showing' },
        { text: 'Coming Soon' }
      ]
    },
    {
      text: 'Directory',
      items: [
        { text: 'Media Gallery' },
        { text: 'Newsletters' }
      ]
    },
    {
      text: 'Queries',
      items: [
        { text: 'Our Policy' },
        { text: 'Site Map' }
      ]
    },
    { text: 'Services' }
  ];
};

```

```

public hiddenItems: string[] = ['Workshops', 'Music', 'Movies'];
public beforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
    //Handling sub menu items
    for (let i: number = 0; i < args.items.length; i++) {
        if (this.hiddenItems.indexOf(args.items[i].text as string) > -1)
        {
            this.menuObj?.hideItems([args.items[i].text as string],
false);
        }
    }
}
public created(): void {
    // Disable menu items
    this.menuObj?.hideItems(this.hiddenItems, false);
}
public btnClick(): void {
    // Show menu items
    this.menuObj?.showItems(this.hiddenItems, false);
    this.hiddenItems = [];
}
}

```

MAIN.TS

```

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

```

Using the [beforeOpen](#) event, you can hide the sub menu items as in the above example since the menu supports to hide items only for headers initially.

Create mnemonic ui in menuitem in Angular Menu component

A particular character in a text can be underlined in the [beforeItemRender](#) event by adding `<u>` tag in between the text and assign the innerHTML to the `li` element.

In the following example, the first character in `File`, `Open`, and `Save` menu items are underlined.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <ejs-menu #menu [items]='menuItems'
    (beforeItemRender)='beforeItemRender($event)'></ejs-menu></div>`
})

```

```

    })
    export class AppComponent {
        public menuItems: MenuItemModel[] = [
            {
                text: 'File',
                iconCss: 'em-icons e-file',
                items: [
                    { text: 'Open', iconCss: 'em-icons e-open' },
                    { text: 'Save', iconCss: 'em-icons e-save' },
                    { separator: true },
                    { text: 'Exit' }
                ]
            },
            {
                text: 'Edit',
                iconCss: 'em-icons e-edit',
                items: [
                    { text: 'Cut', iconCss: 'em-icons e-cut' },
                    { text: 'Copy', iconCss: 'em-icons e-copy' },
                    { text: 'Paste', iconCss: 'em-icons e-paste' }
                ]
            },
            { text: 'Format' },
            { text: 'View' },
            { text: 'Bookmarks' },
            { text: 'Tools' },
            { separator: true },
            { text: 'Help' }
        ];
        public beforeItemRender(args: MenuEventArgs): void {
            if (['File', 'Open', 'Save'].indexOf(args.item.text as string) > -1)
            {
                // To underline a First character.
                let underlinedText: string = '<u>' + (args.item.text as
string).slice(0, 1) + '</u>' + (args.item.text as string).slice(1);
                args.element.innerHTML =
args.element.innerHTML.replace(args.item.text as string, underlinedText);
            }
        }
    }
}

```

MAIN.TS

```

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

```

Change sub menu position in Angular Menu component

The submenu position can be changed by using the [beforeOpen](#) event. Assign the top and left position where you want to open the submenu to the [beforeOpen](#) event arguments `args.top` and `args.left` respectively.

In the below sample, the sub menu opens above the parent menu item.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple, closest } from '@syncfusion/ej2-base';
import { MenuItemModel, BeforeOpenCloseMenuEventArgs } from '@syncfusion/ej2-
angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems'
(beforeOpen)='onBeforeOpen($event)'></ejs-menu>
    </div>`
})
export class AppComponent {
  menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' }
      ]
    },
    {
      text: 'Tools',
      items: [
        { text: 'Spelling & Grammar' },
        { text: 'Customize' },
        { text: 'Options' }
      ]
    },
    { text: 'Go' },
    { text: 'Help' }];
  onBeforeOpen(args: BeforeOpenCloseMenuEventArgs): void {
    // Getting parent menu item element offset

```

```

    let relativeOffset: ClientRect = closest(args.event.target as
Element, '.e-menu-item').getBoundingClientRect();
    // Getting sub menu wrapper element using closest method
    let subMenuEle: HTMLElement = closest(args.element, '.e-menu-
wrapper') as HTMLElement;
    subMenuEle.style.display = 'block';
    args.top = (relativeOffset.top -
subMenuEle.getBoundingClientRect().height) + pageYOffset;
    args.left = relativeOffset.left + pageXOffset;
    subMenuEle.style.display = '';
  }
}

```

MAIN.TS

```

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

```

> For custom positioning, set both **top** and **left** position in the [beforeOpen](#) event.

Rounded corner in Angular Menu component

The rounded corner can be achieved by using the [cssClass](#) property. Add a custom class to the menu component and customize it using the **border-radius** CSS property. For more information, refer to the **style.css** file mapped under the source tab.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems' cssClass='e-rounded-menu'></ejs-
menu>
    </div>`
})
export class AppComponent {
  menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {

```

```

        text: 'Edit',
        items: [
            { text: 'Cut' },
            { text: 'Copy' },
            { text: 'Paste' }
        ]
    },
    {
        text: 'View',
        items: [
            { text: 'Toolbar' },
            { text: 'Sidebar' }
        ]
    },
    {
        text: 'Tools',
        items: [
            { text: 'Spelling & Grammar' },
            { text: 'Customize' },
            { text: 'Options' }
        ]
    },
    { text: 'Go' },
    { text: 'Help' }
];

```

MAIN.TS

```

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

```

Change animation settings in Angular Menu component

To change the animation of the Menu, [animationSettings](#) property is used. The supported effects for Menu are,

| Effect | Functionality |

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

| None | Specifies the sub menu transform with no animation effect. |

| SlideDown | Specifies the sub menu transform with slide down effect. |

| ZoomIn | Specifies the sub menu transform with zoom in effect. |

| FadeIn | Specifies the sub menu transform with fade in effect. |

The following sample illustrates how to open Menu with **FadeIn** [effect](#) with the [duration](#) of **800ms**. Also we can set [easing](#) for menu items.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';

```

```

import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel, MenuAnimationSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems'
[animationSettings]='animationSettings'></ejs-menu></div>`
})
export class AppComponent {
  public animationSettings: MenuAnimationSettingsModel = {
    effect: 'FadeIn',
    duration: 800
  }
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' }
      ]
    },
    {
      text: 'Tools',
      items: [
        { text: 'Spelling & Grammar' },
        { text: 'Customize' },
        { text: 'Options' }
      ]
    },
    { text: 'Go' },
    { text: 'Help' }
  ];
}

```

MAIN.TS

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

Menu item click in Angular Menu component

You can open menu items and sub menu on menu item click by setting [showItemOnClick](#) property of the Menu. To open sub menu items only on item click, should be set as `true`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MenuModule } from '@syncfusion/ej2-angular-navigations'
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel, MenuAnimationSettingsModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems' showItemOnClick = "true"></ejs-
menu></div>`
})
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        {
          text: 'Toolbars',
          items: [
            { text: 'Menu Bar' },
            { text: 'Bookmarks Toolbar' },

```



```

        { text: 'Customize' },
      ],
    },
    {
      text: 'Zoom',
      items: [
        { text: 'Zoom In' },
        { text: 'Zoom Out' },
        { text: 'Reset' },
      ],
    },
    { text: 'Full Screen' }
  ],
},
{
  text: 'Tools',
  items: [
    { text: 'Spelling & Grammar' },
    { text: 'Customize' },
    { text: 'Options' }
  ],
},
{ text: 'Go' },
{ text: 'Help' }
];
}

```

MAIN.TS

```

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

```

Right to left in Angular Menu component

Menu component has RTL support. This can be achieved by setting [enableRtl](#) as `true`.

The following example illustrates how to enable right-to-left support in Menu component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
import { enableRipple } from '@syncfusion/ej2-base';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <!-- To Render Menu. -->
    <ejs-menu [items]='menuItems' enableRtl="true"></ejs-menu></div>`
})

```

```
export class AppComponent {
  public menuItems: MenuItemModel[] = [
    {
      text: 'File',
      items: [
        { text: 'Open' },
        { text: 'Save' },
        { text: 'Exit' }
      ]
    },
    {
      text: 'Edit',
      items: [
        { text: 'Cut' },
        { text: 'Copy' },
        { text: 'Paste' }
      ]
    },
    {
      text: 'View',
      items: [
        { text: 'Toolbar' },
        { text: 'Sidebar' }
      ]
    },
    {
      text: 'Tools',
      items: [
        { text: 'Spelling & Grammar' },
        { text: 'Customize' },
        { text: 'Options' }
      ]
    },
    { text: 'Go' },
    { text: 'Help' }
  ];
}
```

MAIN.TS

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

Set title icon in Angular Menu component

In this sample, the title for settings icon can be achievable by using 'beforeItemRender' client-side event in Menu component.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { MenuModule } from '@syncfusion/ej2-angular-navigations';
import { Component } from '@angular/core';
```

```

import { enableRipple, closest } from '@syncfusion/ej2-base';
import { MenuItemModel, MenuEventArgs } from '@syncfusion/ej2-angular-navigations';
enableRipple(true);
@Component({
  imports: [ MenuModule],
  standalone: true,
  selector: 'app-root',
  template: `<div class="e-section-control">
    <div class="menu-section">
      <div class="menu-control">
        <ejs-menu [items]='menuItems'
        (beforeItemRender)='onBeforeItemRender($event)'>></ejs-menu>
      </div>
    </div>
  </div>`
})
export class AppComponent {

  public menuItems: MenuItemModel[] = [
    {
      id: 'settingIcon',
      iconCss: 'em-icons e-file',
      items: [
        { text: 'Open',
          items: [
            { text: 'Sub Option1' },
            { text: 'Sub Option2' },
          ]
        },
        { text: 'Save' },
        { separator: true },
        { text: 'Exit' }
      ]
    }
  ];
  public onBeforeItemRender(args: MenuEventArgs): void {
    if (args.item.id == 'settingIcon') {
      args.element.setAttribute('title', 'Settings');
    }
  }
}

```

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 Menu component

This article describes the API migration process of Menu component from Essential JS 1 to Essential JS 2.

Properties

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Animation type on hover or click on the menu items | **Property:** *animationType*

 <ej-menu id="menu" [animationType]="animationType"></ej-menu>
 animationType: any = ej.AnimationType.Default; | Not applicable |

| Context menu target | **Property:** *contextMenuTarget*

 <ej-menu id="menu" contextMenuTarget="#target"></ej-menu> | Not applicable |

| Container element for submenu's collision | **Property:** *container*

 <ej-menu id="menu" contextMenuTarget="#target" container="#target"></ej-menu> | Not applicable |

| Menu items | Not applicable | **Property:** *items*

 <ejs-menu id="menu" [items]="menuItems"></ejs-menu>
 menuItems: MenuItemModel[] = [
 {
 text: "File", id: "fileid"
 },
 {
 text: "Edit",
 id: "editid",
 items: [
 { text: "Cut", iconCss: "e-icons e-cut" },
 { text: "Copy", iconCss: "e-icons e-copy" },
 { text: "Paste", iconCss: "e-icons e-paste" }
]
 },
 {
 text: "view", id: "viewid", url: "#"
 },
 text: "Help", id: "helpid"
 }
]; |

| Adding custom class | **Property:** *cssClass*

 <ej-menu id="menu" cssClass="custom-class"></ej-menu> | **Property:** *cssClass*

 <ejs-menu id="menu" cssClass="custom-class" [items]="menuItems"></ejs-menu> |

| Enables or disables the animation on hover or click on the menu items | **Property:** *enableAnimation*

 <ej-menu id="menu" [enableAnimation]="true"></ej-menu> | Not applicable |

| Animation settings | Not applicable | **Property:** *animationSettings*

 <ejs-menu id="menu" [animationSettings]="animation" [items]="menuItems"></ejs-menu>
 animation: MenuAnimationSettings = { effect: "FadeIn" } |

| Root menu items to be aligned center in horizontal menu | **Property:** *enableCenterAlign*

 <ej-menu id="menu" [enableCenterAlign]="true"></ej-menu> | Not applicable |

| Disabled state | **Property:** *enabled*

 <ej-menu id="menu" [enabled]="false"></ej-menu> | **Property:** *disabled*

 <ej-menu id="menu" [disabled]="true" [items]="menuItems"></ej-menu> |

| RTL | **Property:** *enableRTL*

 <ej-menu id="menu" [enableRTL]="true"></ej-menu> | **Property:** *enableRtl*

 <ejs-menu id="menu" [items]="menuItems" [enableRtl]="true"></ejs-menu> |

| Enables/Disables the separator | **Property:** *enableSeparator*

 <ej-menu id="menu" [enableSeparator]="false"></ej-menu> | Not applicable |

| Menu Type | **Property:** *menuType*

 <ej-menu id="menu" [menuType]="menuType"></ej-menu>
 menuType: any = ej.MenuType.ContextMenu; | Not applicable |

| Exclude target for context menu | **Property:** *excludeTarget*

 <ej-menu id="menu" [menuType]="menuType" contextMenuTarget="#target" excludeTarget=".inner"></ej-menu> | Not applicable |

| Fields | **Property:** *fields*

 <ej-menu id="menu" [field]="menuFields">/ej-menu>
 menuFields: Object = { dataSource: this.data, parentId: "parentId", id: "id", text: "text" }; | **Property:** *fields*

 <ejs-menu id="menu" disabled="true" [items]="menuItems" [fields]="menuFields"></ejs-menu>
 menuFields: Object = { text: ["text"], children: ["children"] }; |

| Height | **Property:** *height*

 <ej-menu id="menu" [height]="50"></ej-menu> | Not applicable |

| Width | **Property:** *width*

 <ej-menu id="menu" [width]="800"></ej-menu> | Not applicable |

| HTML Attributes | **Property:** *htmlAttributes*

 <ej-menu id="menu" [htmlAttribute]="attributes"></ej-menu> | Not applicable |

| Responsive | **Property:** *isResponsive*

 <ej-menu id="menu" [isResponsive]="true"></ej-menu> | Not applicable |

| Show item on click | **Property:** *openOnClick*

 <ej-menu id="menu" [openOnClick]="true"></ej-menu> | **Property:** *showItemOnClick*

 <ejs-menu id="menu" [showItemOnClick]="true" [items]="menuItems"></ejs-menu> |

| Orientation | **Property:** *orientation*

 <ej-menu id="menu" orientation="vertical"></ej-menu> | **Property:** *orientation*

 <ejs-menu id="menu" orientation="Vertical" [items]="menuItems"></ejs-menu> |

| Show root level arrows | **Property:** *showRootLevelArrows*

 <ej-menu id="menu" [showRootLevelArrows]="false"></ej-menu> | Not applicable |

| Show sub level arrows | **Property:** *showSubLevelArrows*

 <ej-menu id="menu" [showSubLevelArrows]="false"></ej-menu> | Not applicable |

| Sub menu direction | **Property:** *subMenuDirection*

 <ej-menu id="menu" [subMenuDirection]="direction"></ej-menu>
 direction: any = ej.Direction.Left; | Not applicable |

| Title | **Property:** *titleText*

 <ej-menu id="menu" titleText="Title of the Menu"></ej-menu> | Not applicable |

| Template | Not applicable | **Property:** *template*

 <ejs-menu id="menu" [items]="data" [fields]="menuFields" template="#menuTemplate"></ejs-menu> |

| Pop up menu height | **Property:** *overflowHeight*

 <ej-menu id="menu" [overflowHeight]="200"></ej-menu> | Not applicable |

Methods

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Disable Method | **Method:** *disable*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.disable(); | Not applicable |

| Disable menu items | **Method:** *disableItem*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.disableItem("File"); | **Method:** *enableItems*

 <ejs-menu
id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public
menu: MenuComponent;
 this.menu.enableItems("File", false) |

| Disable menu items by ID | **Method:** *disableItemByID*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.disableItemByID("fileid"); | **Method:** *enableItems*

 <ejs-
menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public
menu: MenuComponent;
 this.menu.enableItems("fileid", false, true) |

| Enable Method | **Method:** *enable*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.enable(); | Not applicable |

| Enable menu items | **Method:** *enableItem*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.enableItem("File"); | **Method:** *enableItems*

 <ejs-menu
id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public
menu: MenuComponent;
 this.menu.enableItems("File", true); |

| Enable menu items by ID | **Method:** *enableItemByID*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.enableItemByID("fileid"); | **Method:** *enableItems*

 <ejs-
menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public
menu: MenuComponent;
 this.menu.enableItems("fileid", true, true); |

| Hide Method | **Method:** *hide*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.hide(); | Not applicable |

| Hide menu items | **Method:** *hideItems*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.hideItems(["#helpid", "#editid"]); | **Method:** *hideItems*

<ejs-menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')

 public menu: MenuComponent;
 this.menu.hideItems(["editid", "helpid"], true); |

| Insert menu items | **Method:** *insert*

 <ej-menu id="menu" #menu
[fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu:
MenuComponent;
 this.menu.insert([{ id: "viewid", text: "View" }], "#editid"); | Not applicable |

| Insert menu items after the specified menu item | **Method:** *insertAfter*

 <ej-menu
id="menu" #menu [fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public
menu: MenuComponent;
 this.menu.insertAfter([{ id: "viewid", text: "View" }], "#editid"); |
Method: *insertAfter*

 <ejs-menu id="menu" #menu [items]="menuItems"></ejs-menu>

 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.insertAfter({ id: "view_id", text: "View" }, "Edit"); |

| Insert menu items before the specified menu item | **Method:** *insertBefore*

 <ej-menu id="menu" #menu [fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.insertBefore({ id: "view_id", text: "View" }, "#helpid"); | **Method:** *insertBefore*

 <ejs-menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.insertBefore({ id: "view_id", text: "View" }, "Help"); |

| Remove menu items | **Method:** *remove*

 <ej-menu id="menu" #menu [fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.remove(["#Edit"]); | **Method:** *removeItems*

 <ejs-menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.removeItems(["Edit"]); |

| To show menu | **Method:** *show*

 <ej-menu id="menu" #menu [fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.show(); | Not applicable |

| Destroy method | **Method:** *destroy*

 <ej-menu id="menu" #menu [fields.dataSource]="data"></ej-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.destroy(); | **Method:** *destroy*

 <ejs-menu id="menu" #menu [items]="menuItems"></ejs-menu>
 @ViewChild('menu')
 public menu: MenuComponent;
 this.menu.destroy(); |

Events

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Triggers before opening the menu | **Events:** *beforeOpen*

 <ej-menu id="menu" (beforeOpen)="beforeOpen(\$event)"></ej-menu>
 beforeOpen(args) {
 / code block /
 } | **Events:** *beforeOpen*

 <ejs-menu id="menu" [items]="menuItems" (beforeOpen)="beforeOpen(\$event)"></ejs-menu>
 beforeOpen(args) {
 / code block /
 } |

| Triggers before closing the menu | Not applicable | **Events:** *beforeClose*

 <ejs-menu id="menu" [items]="menuItems" (beforeClose)="beforeClose(\$event)"></ejs-menu>
 beforeClose(args) {
 / code block */
 } |

| Triggers before rendering each menu item | Not applicable | **Events:** *beforeItemRender*

 <ejs-menu id="menu" [items]="menuItems" (beforeItemRender)="beforeItemRender(\$event)"></ejs-menu>
 beforeItemRender(args) {
 / code block */
 } |

| Triggers while selecting the menu item | **Events:** *click*

 <ej-menu id="menu" (click)="click(\$event)"></ej-menu>
 click(args) {
 / code block /
 } | **Events:** *select*

 <ejs-menu id="menu" [items]="menuItems" (select)="select(\$event)"></ejs-menu>
 select(args) {
 / code block /
 } |

| Triggers after closing the menu | **Events:** `close`
 `<ej-menu id="menu" (close)="close($event)"></ej-menu>`
 `close(args) {`
 `code block`
 `/` `</br>` `}` | **Events:** `onClose`
 `<ej-menu id="menu" [items]="menuitems" (onClose)="onClose($event)"></ej-menu>`
 `onClose(args) {`
 ` / code block`
 `/` `</br>` `}` |

| Triggers after opening the menu | **Events:** `open`
 `<ej-menu id="menu" (open)="open($event)"></ej-menu>`
 `open(args) {`
 ` / code block`
 `/` `</br>` `}` | **Events:** `onOpen`
 `<ej-menu id="menu" [items]="menuitems" (onOpen)="onOpen($event)"></ej-menu>`
 `onOpen(args) {`
 ` / code block`
 `/` `</br>` `}` |

| Triggers once the component rendering is completed | **Events:** `create`
 `<ej-menu id="menu" (create)="create($event)"></ej-menu>`
 `create(args) {`
 ` / code block`
 `/` `</br>` `}` | **Events:** `created`
 `<ej-menu id="menu" [items]="menuitems" (created)="created()"></ej-menu>`
 `created()`
 `{`
 ` / code block`
 `/` `</br>` `}` |

| Triggers once the component is destroyed | **Events:** `destroy`
 `<ej-menu id="menu" (destroy)="destroy($event)"></ej-menu>`
 `destroy(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

| Triggers when key down on menu items | **Events:** `keydown`
 `<ej-menu id="menu" (keydown)="keydown($event)"></ej-menu>`
 `keydown(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

| Triggers when mouse out from menu items | **Events:** `mouseout`
 `<ej-menu id="menu" (mouseout)="mouseout($event)"></ej-menu>`
 `mouseout(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

| Triggers when mouse over the Menu items | **Events:** `mouseover`
 `<ej-menu id="menu" (mouseover)="mouseover($event)"></ej-menu>`
 `mouseover(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

| Triggers when overflow popup menu opens | **Events:** `overflowOpen`
 `<ej-menu id="menu" (overflowOpen)="overflowOpen($event)"></ej-menu>`
 `overflowOpen(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

| Triggers when overflow popup menu closes | **Events:** `overflowClose`
 `<ej-menu id="menu" (overflowClose)="overflowClose($event)"></ej-menu>`
 `overflowClose(args) {`
 ` / code block` `*`
 `</br>` `}` | Not applicable |

Message

Getting started with Angular Message component

This section explains the steps required to create a simple Angular Message component and configure its available functionalities.

Setup Angular Environment

Use [Angular CLI](#) to setup the Angular applications. To install Angular CLI, use the following command.

,


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

```
,
```

Create an Angular Application

Start a new Angular application using the following Angular CLI command.

```
,
```

```
ng new my-app
```

```
cd my-app
```

```
,
```

Installing Syncfusion Notification package

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

Currently, Syncfusion provides two types of package structures for Angular components. They are:

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

Ivy library distribution package

Syncfusion Angular packages($\geq 20.2.36$) has been moved to the Ivy distribution to support the Angular [Ivy](#) rendering engine and the package are compatible with Angular version 12 and above. To download the package, use the following command.

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-notifications --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-notifications@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-notifications:"20.3.47-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 Message module

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

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// Import the Message Module for the Message component
import { MessageModule } from '@syncfusion/ej2-angular-notifications';
import { AppComponent } from './app.component';
@NgModule({
  //Declaration of the MessageModule module into NgModule
  imports: [ BrowserModule, MessageModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
`
```

Adding CSS Reference

Add Message component's styles as given in the following `styles.css`.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-notifications/styles/message/material.css';
`
```

Add Message component

Modify the template in the [src/app/app.component.ts] file to render the Message component. Add the Angular Message by using the `<ejs-message>` selector in the `template` section of the app.component.ts file.

```
`typescript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  template: '<ejs-message content="Please read the comments carefully"></ejs-message>'
})
```

```
export class AppComponent{
}
`
```

Run the application

Use the following command to run the application in the browser.

```
`bash
ng serve --open
`
```

The output will appear as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: '<ejs-message content="Please read the comments
carefully"></ejs-message>'
})
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));
```

Severities in Angular Message component

The severity denotes the importance and context of the message to the user. The message contains different severity types. Use the [severity](#) property to display the messages with different severity levels.

The available severity types are **Normal**, **Success**, **Info**, **Warning** and **Error**. The default severity type for messages is **Normal**.

The following example demonstrates the severity of the messages.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
```

```

imports: [
    MessageModule
],
standalone: true,
selector: 'app-root',
template: `<div class="msg-default-section">
    <div class="content-section">
        <ejs-message id="msg_default" content="Editing is
restricted"></ejs-message>
        <ejs-message id="msg_info" content="Please read the comments
carefully" severity="Info"></ejs-message>
        <ejs-message id="msg_success" content="Your message has been sent
successfully" severity="Success"></ejs-message>
        <ejs-message id="msg_warning" content="There was a problem with
your network connection" severity="Warning"></ejs-message>
        <ejs-message id="msg_error" content="A problem occurred while
submitting your data" severity="Error"></ejs-message>
    </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));

```

Variants in Angular Message component

The Message has predefined appearance variants for different visual representations. The variants of the message can be changed based on the [variant](#) property.

The available variants are **Text**, **Outlined** and **Filled**. The default variant type for messages is **Text**.

- **Text** - The severity is differentiated using a text color and a light background color.
- **Outlined** - The severity is differentiated using a text color and a border without a background.
- **Filled** - The severity is differentiated using a text color and a dark background color.

The following example demonstrates the default message with different variant types.

APP.COMPONENT.TS

```

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

```

```

    template: `<div class="msg-variant-section">
      <div class="content-section">
        <h4>Filled</h4>
        <ejs-message id="msg_default_filled" variant="Filled">Editing is
restricted</ejs-message>
        <ejs-message id="msg_info_filled" severity="Info"
variant="Filled">Please read the comments carefully
        </ejs-message>
        <ejs-message id="msg_success_filled" severity="Success"
variant="Filled">Your message has been sent successfully
        </ejs-message>
        <ejs-message id="msg_warning_filled" severity="Warning"
variant="Filled">There was a problem with your network connection</ejs-
message>
        <ejs-message id="msg_error_filled" severity="Error"
variant="Filled">A problem occurred while submitting your data</ejs-message>
      </div>
      <div class="content-section">
        <h4>Outlined</h4>
        <ejs-message id="msg_default_outlined" variant="Outlined">Editing
is restricted</ejs-message>
        <ejs-message id="msg_info_outlined" severity="Info"
variant="Outlined">Please read the comments carefully</ejs-message>
        <ejs-message id="msg_success_outlined" severity="Success"
variant="Outlined">Your message has been sent successfully</ejs-message>
        <ejs-message id="msg_warning_outlined" severity="Warning"
variant="Outlined">There was a problem with your network connection</ejs-
message>
        <ejs-message id="msg_error_outlined" severity="Error"
variant="Outlined">A problem occurred while submitting your data</ejs-
message>
      </div>
      <div class="content-section">
        <h4>Text</h4>
        <ejs-message id="msg_default">Editing is restricted</ejs-message>
        <ejs-message id="msg_info" severity="Info">Please read the
comments carefully</ejs-message>
        <ejs-message id="msg_success" severity="Success">Your message has
been sent successfully</ejs-message>
        <ejs-message id="msg_warning" severity="Warning">There was a
problem with your network connection</ejs-message>
        <ejs-message id="msg_error" severity="Error">A problem occurred
while submitting your data</ejs-message>
      </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));

```

Icons in Angular Message component

This section explains the message with no icons, how to show or hide the close icon and add the custom severity icon to the message.

No Icon

By default, severity icons can be displayed according to the severity types to make it more understandable to the user by visual information rather than text. To hide the severity icons, set the `showIcon` property to `false`.

The following example demonstrates the different severity messages without the severity icons.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-default-section">
    <div class="content-section">
      <ejs-message id="msg_default" content="Editing is restricted"
[showIcon]="false"></ejs-message>
      <ejs-message id="msg_info" content="Please read the comments
carefully" severity="Info" [showIcon]="false"></ejs-message>
      <ejs-message id="msg_success" content="Your message has been sent
successfully" severity="Success" [showIcon]="false"></ejs-message>
      <ejs-message id="msg_warning" content="There was a problem with
your network connection" severity="Warning" [showIcon]="false"></ejs-message>
      <ejs-message id="msg_error" content="A problem occurred while
submitting your data" severity="Error" [showIcon]="false"></ejs-message>
    </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));
```

Custom Icon

By default, the severity icons can be displayed according to the severity type to make it more understandable to the user by visual information rather than text. If the user wants to customize these icons, it can be achieved through the `cssClass` property.

The following example demonstrates how the default message is rendered with a custom severity icon.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-custom-section">
    <div class="content-section">
      <ejs-message id="msg_icon" cssClass="custom">Essential JS 2 is a
modern JavaScript UI Controls library that has
      been built from the ground up to be lightweight, responsive,
modular and touch friendly. It is written in TypeScript and has no
      external dependencies. It also includes complete support for
Angular, React, Vue, ASP.NET MVC and ASP.NET
      Core frameworks.</ejs-message>
    </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));
```

Close Icon

The message can be rendered with or without the close icon. The close icon is used to hide the message, either by manually clicking the close icon or through keyboard interaction.

By default, the close icon is not rendered in the message. To show the close icon, set the [showCloseIcon](#) property to **true**.

In the following example, the messages are rendered with the close icon.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { Button, ButtonComponent, ChangeEventArgs } from '@syncfusion/ej2-
angular-buttons';
import { Message, MessageComponent } from '@syncfusion/ej2-angular-
notifications';
import { getComponent } from '@syncfusion/ej2-base';
@Component({
```

```

imports: [
    MessageModule, ButtonModule
],
standalone: true,
selector: 'app-root',
template: `<div class="msg-icon-section">
    <div class="content-section">
        <button #btn1 ejs-button content="Show Default Message"
cssClass="e-outline e-primary msg-hidden" (click)="defaultClick()"></button>
        <ejs-message #msg_default_icon id="msg_default_icon"
[showCloseIcon]="true" (closed)="defaultClosed()">Editing is restricted</ejs-
message>
        <button #btn2 ejs-button content="Show Info Message" cssClass="e-
outline e-primary e-info msg-hidden" (click)="infoClick()"></button>
        <ejs-message #msg_info_icon id="msg_info_icon" severity="Info"
[showCloseIcon]="true" (closed)="infoClosed()">Please read the comments
carefully</ejs-message>
        <button #btn3 ejs-button content="Show Success Message"
cssClass="e-outline e-primary e-success msg-hidden"
(click)="successClick()"></button>
        <ejs-message #msg_success_icon id="msg_success_icon"
severity="Success" [showCloseIcon]="true" (closed)="successClosed()">Your
message has been sent successfully</ejs-message>
        <button #btn4 ejs-button content="Show Warning Message"
cssClass="e-outline e-primary e-warning msg-hidden"
(click)="warningClick()"></button>
        <ejs-message #msg_warning_icon id="msg_warning_icon"
severity="Warning" [showCloseIcon]="true" (closed)="warningClosed()">There
was a problem with your network connection</ejs-message>
        <button #btn5 ejs-button content="Show Error Message"
cssClass="e-outline e-primary e-error msg-hidden"
(click)="errorClick()"></button>
        <ejs-message #msg_error_icon id="msg_error_icon" severity="Error"
[showCloseIcon]="true" (closed)="errorClosed()">A problem has been occurred
while submitting your data</ejs-message>
    </div>
</div>`
})
export class AppComponent{
    @ViewChild('btn1') private defaultBtn?: ButtonComponent;
    @ViewChild('btn2') private infoBtn?: ButtonComponent;
    @ViewChild('btn3') private successBtn?: ButtonComponent;
    @ViewChild('btn4') private warningBtn?: ButtonComponent;
    @ViewChild('btn5') private errorBtn?: ButtonComponent;
    @ViewChild('msg_default_icon') private msgDefault?: MessageComponent;
    @ViewChild('msg_success_icon') private msgSuccess?: MessageComponent;
    @ViewChild('msg_warning_icon') private msgWarning?: MessageComponent;
    @ViewChild('msg_error_icon') private msgError?: MessageComponent;
    @ViewChild('msg_info_icon') private msgInfo?: MessageComponent;
    public defaultClick(): void {
        this.show(this.msgDefault as MessageComponent, this.defaultBtn as
ButtonComponent);
    }
    public defaultClosed(): void {
        this.defaultBtn?.element.classList.remove('msg-hidden');
    }
    public infoClick(): void {

```



```

        this.show(this.msgInfo as MessageComponent, this.infoBtn as
        ButtonComponent);
    }
    public infoClosed(): void {
        this.infoBtn?.element.classList.remove('msg-hidden');
    }
    public successClick(): void {
        this.show(this.msgSuccess as MessageComponent, this.successBtn as
        ButtonComponent);
    }
    public successClosed(): void {
        this.successBtn?.element.classList.remove('msg-hidden');
    }
    public warningClick(): void {
        this.show(this.msgWarning as MessageComponent, this.warningBtn as
        ButtonComponent);
    }
    public warningClosed(): void {
        this.warningBtn?.element.classList.remove('msg-hidden');
    }
    public errorClick(): void {
        this.show(this.msgError as MessageComponent, this.errorBtn as
        ButtonComponent );
    }
    public errorClosed(): void {
        this.errorBtn?.element.classList.remove('msg-hidden');
    }
    public show(message: Message, btn: Button): void {
        message.visible = true;
        btn.element.classList.add('msg-hidden');
    }
}

```

MAIN.TS

```

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

```

Customization in Angular Message component

The Message component allows the user to customize the content display positions and appearance. This section explains the details about changing the content alignments and border styles for messages.

Content Alignment

Normally, the message content is aligned to the **left**. The Message component allows the user to align the message content in the **center** or **right** through the built-in classes **e-content-center** and **e-content-right**.

The following example demonstrates the message with different content alignments.

APP.COMPONENT.TS

```

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

```

```
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-custom-section">
    <div class="content-section">
      <h4>Content Alignment</h4>
      <ejs-message id="msg_content_left" content="Your license has been
activated successfully" severity="Success"></ejs-message>
      <ejs-message id="msg_content_center" content="The license will
expire today" cssClass="e-content-center" severity="Warning"></ejs-message>
      <ejs-message id="msg_content_right" content="The license key is
invalid" cssClass="e-content-right" severity="Error"></ejs-message>
    </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));
```

Rounded and Square

To customize the Message component's appearance, add the custom class to the message through the [cssClass](#) property. This custom class will be added to the root element. Based on this custom class, the user can override the message styles at the application level.

The following example shows the rounded and squared appearance of the message, which can be achieved by adding the [cssClass](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-custom-section">
    <div class="content-section">
      <h4>Rounded</h4>
      <ejs-message content="The license will expire today"
cssClass="rounded" severity="Warning"></ejs-message>
    </div>
  </div>`
})
```

```

        <h4>Square</h4>
        <ejs-message content="The license key is invalid"
cssClass="square" severity="Error"></ejs-message>
    </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));

```

CSS Message

The Essential JS 2 Message has predefined CSS classes that can be defined in the HTML elements, which renders the message without any script reference. This can display a simple message with content and make the code lighter.

The following DOM structure is required to display the simple message with the content.

```

`bash
<div class="e-message">
<div class="e-msg-content">..content..</div>
</div>
`

```

The following DOM structure is required to display the simple message with the content and severity icon.

```

`bash
<div class="e-message">
<span class="e-msg-icon"></span>
<div class="e-msg-content">..content..</div>
</div>
`

```

The following is the available list of predefined CSS classes to make the appearance of a message.

Class	Description
-----	-----
e-message	Represents the message wrapper.
e-msg-icon	Represents the severity type icon.
e-msg-content	Represents the message content.

- | e-msg-close-icon | Represents the close icon. |
- | e-info | Represents the information message. |
- | e-success | Represents the success message. |
- | e-warning | Represents the warning message. |
- | e-error | Represents the error message. |
- | e-content-center | Aligns the message content to the center. |
- | e-content-right | Aligns the message content to the right. |

The following example shows the message which renders without any script reference.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { Component } from '@angular/core';
@Component({
  imports: [
    MessageModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-default-section">
    <div class="content-section">
      <div id="msg-default" class="e-message" role="alert">
        <span class="e-msg-icon"></span>
        <div class="e-msg-content">Editing is restricted</div>
      </div>
      <div id="msg-info" class="e-message e-info" role="alert">
        <span class="e-msg-icon"></span>
        <div class="e-msg-content">Please read the comments
carefully</div>
      </div>
      <div id="msg-success" class="e-message e-success" role="alert">
        <span class="e-msg-icon"></span>
        <div class="e-msg-content">Your message has been sent
successfully</div>
      </div>
      <div id="msg-warning" class="e-message e-warning" role="alert">
        <span class="e-msg-icon"></span>
        <div class="e-msg-content">There was a problem with your
network connection</div>
      </div>
      <div id="msg-error" class="e-message e-error" role="alert">
        <span class="e-msg-icon"></span>
        <div class="e-msg-content">A problem occurred while
submitting your data</div>
      </div>
    </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));
```

Template in Angular Message component

The message supports templates that allows the user to customize the content with a custom structure. The content can be a string, paragraph, or any other HTML element. The template can be rendered through the [content](#) property or added directly to the HTML element.

In the following sample, the Message component content is customized with HTML elements and Angular Button components, which are directly added to the HTML element.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { MessageModule } from '@syncfusion/ej2-angular-notifications'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, ViewChild, ViewEncapsulation } from '@angular/core';
import { ButtonComponent } from '@syncfusion/ej2-angular-buttons';
import { MessageComponent } from '@syncfusion/ej2-angular-notifications';
@Component({
  imports: [
    MessageModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<div class="msg-template-section">
    <div class="content-section">
      <button ej2-button #showBtn id='showBtn' content='Show pull
request' cssClass="e-outline e-primary e-success msg-hidden"
(click)="showClick()"></button>
      <ejs-message #msg_template id="msg_template" severity="Success"
(closed)="closed()">
        <ng-template #content>
          <h1>Merged pull request</h1>
          <p>Pull request #41 merged after a successful build</p>
          <button ej2-button id='commitBtn' cssClass='e-link'
content='View commit'></button>
          <button ej2-button #closeBtn id='closeBtn' cssClass='e-
link' content='Dismiss' (click)="dismissClick()"></button>
        </ng-template>
      </ejs-message>
    </div>
  </div>`
})
export class AppComponent{
  @ViewChild('showBtn') private showBtn?: ButtonComponent;
  @ViewChild('msg_template') private msgTemplate?: MessageComponent;
  public showClick(): void {
    this.msgTemplate!.visible = true;
```

```

        this.showBtn?.element.classList.add('msg-hidden');
    }
    public dismissClick(): void {
        this.msgTemplate!.visible = false;
    }
    public closed(): void {
        this.showBtn?.element.classList.remove('msg-hidden');
    }
}

```

MAIN.TS

```

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

```

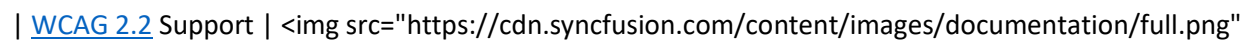
Accessibility in Angular Message component

The Message 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 Message component is outlined below.

| Accessibility Criteria | Compatibility |


| -- | -- |

| [WCAG 2.2](#) Support |  alt="Yes" > |

| [Section 508](#) Support |  alt="Yes" > |

| Screen Reader Support |  alt="Yes" > |

| Right-To-Left Support |  alt="Yes" > |

| Color Contrast |  alt="Yes" > |

| Mobile Device Support |  alt="Yes" > |

| Keyboard Navigation Support |  alt="Yes" > |

| [Accessibility Checker](#) Validation |  alt="Yes" > |

| [Axe-core](#) Accessibility Validation |  alt="Yes" > |

<style>

.post .post-content img {

```
display: inline-block;
margin: 0.5em 0;
}
```

```
</style>
```

```
<div> - 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 Message component followed the [WAI-ARIA](#) patterns to meet the accessibility. The following ARIA attributes are used in the Message component:

Attributes	Purpose
------------	---------

---	---
-----	-----

role=alert	Used to convey a significant and contextual message to the user.
-------------------	------------------------------------------------------------------

aria-label	Provides an accessible name for the close icon.
-------------------	-------------------------------------------------

Keyboard interaction

The Message component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Message component.

Press	To do this
-------	------------

---	---
-----	-----

Tab / Shift + Tab	To focus the close icon in the message.
--------------------------	-----------------------------------------

Enter / Space	Closes the focused close icon's message.
----------------------	------------------------------------------

Ensuring accessibility

The Message component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the Message component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the Message component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

MultiSelect

Getting started with Angular Multi select component

This section explains how to create a simple **MultiSelect** component and configure its available functionalities in Angular.

Dependencies

The following list of dependencies are required to use the Angular MultiSelect component in your application.

```
`javascript
|-- @syncfusion/ej2-angular-dropdowns
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-data
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-dropdowns
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-lists
|-- @syncfusion/ej2-navigations
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
`,`
```

Setup angular environment

Angular provides the easiest way to set angular CLI projects using [Angular CLI](#) tool.

Install the CLI application globally to your machine.

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

Create a new application

```
`bash
ng new syncfusion-angular-multiselect
`,`
```

By default, it install the CSS style base application. To setup with SCSS, pass `--style=scss` argument on create project.

Example code snippet.

```
`bash
ng new syncfusion-angular-multiselect --style=scss
`,`
```

Navigate to the created project folder.

```
`bash
cd syncfusion-angular-multiselect
`,`
```


Installing Syncfusion MultiSelect 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-dropdowns](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-dropdowns --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-dropdowns@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-dropdowns@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-dropdowns:"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 MultiSelect module

Import MultiSelect module into Angular application(`app.module.ts`) from the package `@syncfusion/ej2-angular-dropdowns`.

```
`javascript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the MultiSelectModule for the MultiSelect component
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
```

```
import { AppComponent } from './app.component';
@NgModule({
  //declaration of ej2-angular-dropdowns module into NgModule
  imports: [ BrowserModule, MultiSelectModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Adding CSS reference

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

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-inputs/styles/material.css';
@import '../node_modules/@syncfusion/ej2-popups/styles/material.css';
@import '../node_modules/@syncfusion/ej2-lists/styles/material.css';
@import '../node_modules/@syncfusion/ej2-angular-dropdowns/styles/material.css';
```

Adding MultiSelect component

Modify the template in `[src/app/app.component.ts]` file to render the Angular MultiSelect component. Add the Angular MultiSelect by using `<ejs-multiselect>` selector in `template` section of the `app.component.ts` file.

```
`javascript
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: <ejs-multiselect id='multiselectelement'></ejs-multiselect>
})
export class AppComponent { }
```

Binding data source

After initialization, populate the MultiSelect with data using the `dataSource` property. Here, an array of string values passed to MultiSelect component.

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

@Component({
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: <ejs-multiselect id='multiselectelement' [dataSource]='data'></ejs-multiselect>
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: string[] = ['Badminton', 'Basketball', 'Cricket', 'Football', 'Golf', 'Gymnastics', 'Hockey',
    'Rugby', 'Snooker', 'Tennis'];
}
```

Running the application

After completing the configuration required to render a basic MultiSelect, run the following command to display the output in your default browser.

```
ng serve
```

The following example illustrates the output in your browser.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component with
  dataSource
```

```

    template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
    [placeholder]='placeholder'></ejs-multiselect>`
  })
  export class AppComponent {
    constructor() {
    }
    // defined the array of data
    public data: string[] = ['Badminton', 'Cricket', 'Football', 'Golf',
    'Tennis'];
    // set placeholder to MultiSelect input element
    public placeholder: string = 'Select games';
  }

```

MAIN.TS

```

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

```

Configure the popup list

By default, the width of the popup list automatically adjusts according to the MultiSelect input element's width, and the height auto adjust's according to the height of the popup list items.

The height and width of the popup list can also be customized using the [popupHeight](#)

 and [popupWidth](#) properties respectively.

In the following sample, popup list's width and height are configured.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component with
  dataSource
  template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
  [placeholder]='placeholder' [popupHeight]='popupHeight'
  [popupWidth]='popupWidth'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: string[] = ['Badminton', 'Cricket', 'Football', 'Golf',
  'Hockey', 'Rugby'];

```

```
// set placeholder to MultiSelect input element
public placeholder: string = 'Select games';
//set height to popup list
public popupHeight:string = '200px';
//set width to popup list
public popupWidth:string = '250px';
}
```

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 MultiSelect](#) feature tour page for its groundbreaking feature representations. You can also explore our [Angular MultiSelect example](#) that shows how to render the MultiSelect Dropdown in Angular.

Data binding in Angular Multi select component

The MultiSelect loads the data either from local data sources or remote data services using the [dataSource](#) property. It supports the data type of `array` or `DataManager`.

The MultiSelect also supports different kinds of data services such as OData, OData V4, and Web API, and data formats such as XML, JSON, and JSONP with the help of `DataManager` adaptors.

Fields	Type	Description
text	string	Specifies the display text of each list item.
value	number or string	Specifies the hidden data value mapped to each list item that should contain a unique value.
groupBy	string	Specifies the category under which the list item has to be grouped.
iconCss	string	Specifies the icon class of each list item.

When binding complex data to the MultiSelect, fields should be mapped correctly. Otherwise, the selected item remains undefined.

Binding local data

Local data can be represented in two ways as described below.

1. Array of string

The MultiSelect has support to load array of primitive data such as strings and numbers. Here, both value and text field act the same.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
```

```
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component with
  // dataSource
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [placeholder]='placeholder'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public sportsData: string[] = ['Badminton', 'Cricket', 'Football',
'Golf', 'Tennis'];
  // set placeholder to MultiSelect input element
  public placeholder: string = 'Select games';
}
```

MAIN.TS

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

2. Array of object

The MultiSelect can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, **id** column and **sports** column from complex data have been mapped to the **value** field and **text** field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [placeholder]='placeholder'></ejs-multiselect>`
})
```

```
export class AppComponent {
  constructor() {
    //set the data to dataSource property
    public sportsData: Object[] = [
      { id: 'Game1', sports: 'Badminton' },
      { id: 'Game2', sports: 'Basketball' },
      { id: 'Game3', sports: 'Cricket' },
      { id: 'Game4', sports: 'Football' },
      { id: 'Game5', sports: 'Golf' }
    ];
    // maps the appropriate column to fields property
    public fields: Object = { text: 'sports', value: 'id' };
    // set placeholder to MultiSelect input element
    public placeholder: string = 'Select games';
  }
}
```

MAIN.TS

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

2. Array of complex object

The MultiSelect can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property.

In the following example, `Code.Id` column and `Country.Name` column from complex data have been mapped to the `value` field and `text` field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [placeholder]='placeholder'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
    //set the data to dataSource property
    public sportsData: Object[] = [
```

```

        { Country: { Name: 'Australia' }, Code: { Id: 'AU' } },
        { Country: { Name: 'Bermuda' }, Code: { Id: 'BM' } },
        { Country: { Name: 'Canada' }, Code: { Id: 'CA' } },
        { Country: { Name: 'Cameroon' }, Code: { Id: 'CM' } },
        { Country: { Name: 'Denmark' }, Code: { Id: 'DK' } },
        { Country: { Name: 'France' }, Code: { Id: 'FR' } }
    ];
    // maps the appropriate column to fields property
    public fields: Object = { text: 'Country.Name', value: 'Code.Id' };
    // set placeholder to MultiSelect input element
    public placeholder: string = 'Select a country';
}

```

MAIN.TS

```

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

```

Binding remote data

The MultiSelect supports retrieval of data from remote data services with the help of

DataManager component. The [Query](#) property is used to fetch data from the database and bind it to the MultiSelect.

The following sample displays the first 6 contacts from “Customers” table of the **Northwind** Data Service.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
//import DataManager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
    [fields]='fields' [placeholder]='placeholder' [query]='query'
    [sortOrder]='sorting'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public data: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',

```



```

        adaptor: new ODataV4Adaptor,
        crossDomain: true
    });
    // maps the appropriate column to fields property
    public fields: Object = { text: 'ContactName', value: 'CustomerID' };
    //bind the Query instance to query property
    public query: Query = new
    Query().from('Customers').select(['ContactName', 'CustomerID']).take(5);
    // set placeholder to MultiSelect input element
    public placeholder: string = 'Select customers';
    //sort the result items
    public sorting: string = 'Ascending';
}

```

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 using Async pipe

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

MultiSelect 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.

[app.component.ts]

`ts

```

import { Component } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import { HttpClient } from '@angular/common/http';
@Component({
  selector: 'app-root',
  // specifies the template string for the Multiselect component with dataSource
  template: <ejs-multiselect id='customers2' formControlName="skillname" name="skillname"
  #remote2 [dataSource]='data | async' [fields]='remoteFields'
  [placeholder]='remoteWaterMark' ></ejs-multiselect >,
})
export class AppComponent {
  constructor(private http: HttpClient){
    this.data=this.http.get<[[key:
    string]:object;]]>('https://services.odata.org/V4/Northwind/Northwind.svc/Customers').pipe(

```

```
map((results : {[key: string]:any;}) => {
return results['value'];
})
);
}

public data: Observable<any>;
// maps the remote data column to fields property
public remoteFields: Object = { value: 'CustomerID' };
// set the placeholder to Multiselect input element
public remoteWaterMark: string = 'Select a customer';
}
`

[app.module.ts]
`ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { DropDownListModule, MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { AppComponent } from './app.component';
import { DialogModule } from '@syncfusion/ej2-angular-popups';
import { ReactiveFormsModule } from '@angular/forms';
@NgModule({
imports: [
BrowserModule,
FormsModule,
DropDownListModule,
MultiSelectModule,
DialogModule,
HttpClientModule,
ReactiveFormsModule
],
declarations: [ AppComponent ],
```

```
bootstrap: [ AppComponent ]
})
export class AppModule { }
`
[main.ts]
`ts
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';
import { AppModule } from './app.module';
enableProdMode();
platformBrowserDynamic().bootstrapModule(AppModule);
`
```

[View Sample in Github](#)

See Also

- [How to load data using template](#)
- [How to group the data using header](#)
- [How to filter the bound data](#)

Value binding in ##Platform_Name## Multi select control

Value binding in the MultiSelect control allows you to associate data values with each list item. This facilitates managing and retrieving selected values efficiently. The MultiSelect component provides flexibility in binding both primitive data types and complex objects.

Primitive Data Types

The MultiSelect Dropdown control provides flexible binding capabilities for primitive data types like strings and numbers. You can effortlessly bind local primitive data arrays, fetch and bind data from remote sources, and even custom data binding to suit specific requirements. Bind the value of primitive data to the [value](#) property of the MultiSelect.

Primitive data types include:

- String
- Number
- Boolean
- Null

The following sample shows the example for preselect values for primitive data type

APP.COMPONENT.TS

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

```
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: string[] = [];
  constructor() {
    this.records = ["Item 1", "Item 2", "Item 3", "Item 4", "Item 5",
    "Item 6", "Item 7", "Item 8", "Item 9", "Item 10"];
  }
  // maps the appropriate column to fields property
  public fields: object = { text: 'text', value: 'id' };
  public value = ["Item 11", "Item 22", "Item 33"];
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}
```

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.HTML

```
<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [value]='value' popupHeight='200px'
[allowFiltering]='false' [placeholder]='waterMark'>
    </ejs-multiselect>
  </div>
</div>
```

Object Data Types

In the MultiSelect Dropdown control, object binding allows you to bind to a dataset of objects. When [allowObjectBinding](#) is enabled, the value of the control will be an object of the same type as the selected item in the [value](#) property. This feature seamlessly binds arrays of objects, whether sourced locally, retrieved from remote endpoints, or customized to suit specific application needs.

The following sample shows the example for preselect values for object data type

APP.COMPONENT.TS

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

```

import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
        text: `Item ${i}`,
      };
      this.records.push(item);
    }
    // maps the appropriate column to fields property
    public fields: object = { text: 'text', value: 'id' };
    public value = [{id: 'id11', text: 'Item 11'}, {id: 'id22', text: 'Item 22'}, {id: 'id33', text: 'Item 33'}];
    // set the placeholder to AutoComplete input
    public waterMark: string = 'e.g. Item 1';
  }
}

```

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.HTML

```

<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [value]='value' [fields]='fields' popupHeight='200px'
[allowObjectBinding]='true' [allowFiltering]='false'
[placeholder]='waterMark'>
      </ejs-multiselect>
    </div>
  </div>
</div>

```

Templates in Angular Multi select component

The MultiSelect has been provided with several options to customize each list item, group title, selected value, header, and footer elements. It uses the Essential JS 2 **Template engine** to compile and render the elements properly.

Item template

The content of each list item within the MultiSelect can be customized with the help of [itemTemplate](#) property.

In the following sample, each list item is split into two columns to display relevant data's.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'template.html'
})
export class AppComponent {
  height: any;
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public data: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { text: 'FirstName', value: 'EmployeeID' };
  //bind the Query instance to query property
  public query: Query = new Query().from('Employees').select(['FirstName', 'City', 'EmployeeID']).take(6);
  //set the placeholder to MultiSelect input
  public text: string = "Select an employee";
  //sort the result items
  public sorting: string = 'Ascending';
}
```

MAIN.TS

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

TEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the template string for the MultiSelect component-->
    <ejs-multiselect id='multiselect-template' #sample
[dataSource]='data' [fields]='fields' [sortOrder]='sorting' [query]='query'
[popupHeight]='height' [placeholder]='text'>
      <ng-template #itemTemplate="" let-data="">
        <!--set the value to itemTemplate property-->
        <div><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></div>
      </ng-template>
    </ejs-multiselect>
  </div>
</div>

```

Value template

The currently selected value that is displayed by default on the MultiSelect input element can be customized using the [valueTemplate](#) property.

In the following sample, the selected value is displayed as a combined text of both **FirstName** and **City** in the MultiSelect input, which is separated by a hyphen.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule,MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'template.html'
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public data: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { text: 'FirstName', value: 'EmployeeID' };
  //bind the Query instance to query property
  public query: Query = new Query().from('Employees').select(['FirstName',
'City','EmployeeID']).take(6);
  //set the placeholder to MultiSelect input

```

```

    public text: string = "Select an employee";
    //sort the result items
    public sorting: string = 'Ascending';
    public box : string = 'box';
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper">
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the template string for the MultiSelect component-->
    <ejs-multiselect id='multiselectelement' #sample [dataSource]='data'
[fields]='fields' [mode]='box' [sortOrder]='sorting' [placeholder]='text'
[query]='query'>
      <ng-template #itemTemplate="" let-data="">
        <!--set the value to itemTemplate property-->
        <div><span class='name'>{{data.FirstName}}</span><span class
='city'>{{data.City}}</span></div>
      </ng-template>
      <ng-template #valueTemplate="" let-data="">
        <!--set the value to valueTemplate property-->
        <span>{{data.FirstName}} - {{data.City}}</span>
      </ng-template>
    </ejs-multiselect>
  </div>
</div>

```

Group template

The group header title under which appropriate sub-items are categorized can also be customize with the help of [groupTemplate](#) property. This template is common for both inline and floating group header template.

In the following sample, employees are grouped according to their city.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, Predicate, DataManager, ODataV4Adaptor } from
 '@syncfusion/ej2-data'
@Component({
  imports: [
    FormsModule,MultiSelectModule
  ],
  standalone: true,

```



```

    selector: 'app-root',
    // specifies the template url path
    templateUrl: 'template.html'
  })
  export class AppComponent {
    constructor() {
    }
    //bind the DataManager instance to dataSource property
    public data: DataManager = new DataManager({
      url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
      adaptor: new ODataV4Adaptor,
      crossDomain: true
    });
    // form predicate to fetch the grouped data
    public groupPredicate = new Predicate('City',
    'equal', 'london').or('City', 'equal', 'seattle');
    // maps the appropriate column to fields property
    public fields: Object = { text: 'FirstName', value: 'EmployeeID',
    groupBy: 'City' };
    //bind the Query instance to query property
    public query: Query = new Query().from('Employees').select(['FirstName',
    'City', 'EmployeeID']).take(6).where(this.groupPredicate);
    //set the placeholder to MultiSelect input
    public text: string = "Select an employee";
    //sort the result items
    public sorting: string = 'Ascending';
  }

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:250px;">
    <!-- specifies the template string for the MultiSelect component-->
    <ejs-multiselect id='multiselectelement' #sample [dataSource]='data'
    [fields]='fields' [sortOrder]='sorting' [placeholder]='text' [query]='query'>
      <ng-template #groupTemplate="" let-data="">
        <!--set the value to groupTemplate property-->
        <strong>{{data.City}}</strong>
      </ng-template>
    </ejs-multiselect>
  </div>
</div>

```

Header template

The header element is shown statically at the top of the popup list items within the MultiSelect, and any custom element can be placed as a header element using the [headerTemplate](#) property.

In the following sample, the list items and its headers are designed and displayed as two columns similar to multiple columns of the grid.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'template.html'
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public data: DataManager = new DataManager({
    url: 'https://services.odata.org/V4/Northwind/Northwind.svc/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public fields: Object = { text: 'FirstName', value: 'EmployeeID' };
  //bind the Query instance to query property
  public query: Query = new Query().from('Employees').select(['FirstName',
  'City', 'EmployeeID']).take(6);
  //set the placeholder to MultiSelect input
  public text: string = "Select an employee";
  //sort the result items
  public sorting: string = 'Ascending';
}
```

MAIN.TS

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

TEMPLATE.HTML

```
<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:250px;">
    <!-- specifies the template string for the MultiSelect component-->
    <ejs-multiselect id='multiselectelement' #sample [dataSource]='data'
    [fields]='fields' [sortOrder]='sorting' [placeholder]='text' [query]='query'>
      <ng-template #itemTemplate="" let-data="">
        <!--set the value to itemTemplate property-->
```

```

        <span class='item'><span class='name'>
{{data.FirstName}}</span><span class='city'>{{data.City}}</span></span>
        </ng-template>
        <ng-template #headerTemplate="" let-data="">
        <!--set the value to headerTemplate property-->
        <span class='head'><span class='name'>Name</span><span
class='city'>City</span></span>
        </ng-template>
    </ejs-multiselect>
</div>
</div>

```

Footer template

The MultiSelect has options to show a footer element at the bottom of the list items in the popup list. Here, you can place any custom element as a footer element using the [footerTemplate](#) property.

In the following sample, footer element displays the total number of list items present in the MultiSelect.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule,MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template url path
  templateUrl: 'template.html'
})
export class AppComponent {
  // create a object for MultiSelect
  constructor() {
  }
  // defined the array of data
  public data: Object[] = ['Badminton', 'Basketball', 'Cricket', 'Golf'];
  // set placeholder text to MultiSelect input element
  public text: string = 'Select a game';
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>

```

```

<div id='content' style='margin: 0px auto; width:250px;'>
  <!-- specifies the template string for the MultiSelect component-->
  <ejs-multiselect id='multiselectelement' #samples [dataSource]='data'
[placeholder]='text' [footerTemplate]='footerTemplate'>
    <ng-template #footerTemplate="" let-data="">
      <!--set the value to footerTemplate property-->
      <span class='foot'> Total list item: 4</span>
    </ng-template>
  </ejs-multiselect>
</div>
</div>

```

No records template

The MultiSelect is provided with support to custom design the popup list content when no data is found and no matches found on search with the help of [noRecordsTemplate](#) property.

In the following sample, popup list content displays the notification of no data available.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule,MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
placeholder='Find a item'>
    <ng-template #noRecordsTemplate>
      <span class='norecord'> NO DATA AVAILABLE</span>
    </ng-template>
  </ejs-multiselect>`
})
export class AppComponent {
  // defined the empty array data
  public data: string[] = [];
}

```

MAIN.TS

```

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

```

Action failure template

There is also an option to custom design the popup list content when the data fetch request fails at the remote server. This can be achieved using the [actionFailureTemplate](#) property.

In the following sample, when the data fetch request fails, the MultiSelect displays the notification.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
//import data manager related classes
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
[query]='query'
                [fields]='fields' placeholder='Find an employee'>
                <ng-template #actionFailureTemplate>
                  <span class='action-failure'> Data fetch get
fails</span>
                </ng-template>
                </ejs-multiselect>`
})
export class AppComponent {
  //bind the data manager instance to dataSource property
  public data: DataManager = new DataManager({
    // Here, use the wrong url to display the action failure template
    url: 'https://services.odata.org/V4/Northwind/Northwind.svcs/',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  //bind the Query instance to query property
  public query: Query = new
Query().from('Employees').select(['FirstName']).take(6);
  // maps the appropriate column to fields property
  public fields: Object = { text: 'FirstName', value: 'EmployeeID' };
}
```

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 bind the data](#)
- [How to group the data using header](#)
- [How to customize the options in MultiSelect](#)

Grouping in Angular Multi select component

The MultiSelect supports wrapping nested elements into a group based on different categories. The category of each list item can be mapped through the [groupBy](#) field in the data table. The group header is displayed both as inline and fixed headers. The fixed group header content is updated dynamically on scrolling the popup list with its category value.

In the following sample, vegetables are grouped according on its category using `groupBy` field.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='vegetableData' [fields]='fields' [placeholder]='placeholder'
[popupHeight]='height'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  //define the data with category
  public vegetableData: { [key: string]: Object }[] = [
    { vegetable: 'Cabbage', category: 'Leafy and Salad', id: 'item1' },
    { vegetable: 'Spinach', category: 'Leafy and Salad', id: 'item2' },
    { vegetable: 'Wheat grass', category: 'Leafy and Salad', id: 'item3' },
    { vegetable: 'Yarrow', category: 'Leafy and Salad', id: 'item4' },
    { vegetable: 'Pumpkins', category: 'Leafy and Salad', id: 'item5' },
    { vegetable: 'Chickpea', category: 'Beans', id: 'item6' },
    { vegetable: 'Green bean', category: 'Beans', id: 'item7' },
    { vegetable: 'Horse gram', category: 'Beans', id: 'item8' },
    { vegetable: 'Garlic', category: 'Bulb and Stem', id: 'item9' },
    { vegetable: 'Nopal', category: 'Bulb and Stem', id: 'item10' },
    { vegetable: 'Onion', category: 'Bulb and Stem', id: 'item11' }
  ];
  // map the groupBy field with category column
  public fields: Object = { groupBy: 'category', text: 'vegetable', value:
'id' };
  // Set the popup list height
  public height: string = '200px';
  // set the placeholder to the MultiSelect input
  public placeholder: string = 'Select vegetables';
}
```

MAIN.TS

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

Customization

The grouping header is also provided with customization option. This allows custom designing using the [groupTemplate](#) property for both inline and fixed headers.

Grouping with CheckBox

Previously, there is no checkbox for group headers. Now, this feature allow to render checkbox in group header to select the group items in single selection. You can enable this feature by setting [enableGroupCheckBox](#) property value as **true** and **mode** property as **CheckBox**.

Inject the `CheckBoxSelection` module in the MultiSelect to use the checkbox.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { CheckBoxSelectionService } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='vegetableData' [fields]='fields' [placeholder]='placeholder'
[popupHeight]='height' [mode]='mode'
[enableGroupCheckBox]='enableGroupCheckBox' [allowFiltering]='allowFiltering'
[filterBarPlaceholder]='filterBarPlaceholder'
[showSelectAll]='showSelectAll'></ejs-multiselect>`,
  providers: [CheckBoxSelectionService]
})
export class AppComponent {
  public mode?: string;
  constructor() {
  }
  //define the data with category
  public vegetableData: { [key: string]: Object }[] = [
    { vegetable: 'Cabbage', category: 'Leafy and Salad', id: 'item1' },
    { vegetable: 'Spinach', category: 'Leafy and Salad', id: 'item2' },
    { vegetable: 'Wheat grass', category: 'Leafy and Salad', id: 'item3' },
    { vegetable: 'Yarrow', category: 'Leafy and Salad', id: 'item4' },
    { vegetable: 'Pumpkins', category: 'Leafy and Salad', id: 'item5' },
    { vegetable: 'Chickpea', category: 'Beans', id: 'item6' },
    { vegetable: 'Green bean', category: 'Beans', id: 'item7' },
    { vegetable: 'Horse gram', category: 'Beans', id: 'item8' },
  ],
```

```

        { vegetable: 'Garlic', category: 'Bulb and Stem', id: 'item9' },
        { vegetable: 'Nopal', category: 'Bulb and Stem', id: 'item10' },
        { vegetable: 'Onion', category: 'Bulb and Stem', id: 'item11' }
    ];
    // map the groupBy field with category column
    public fields: Object = { groupBy: 'category', text: 'vegetable', value:
'id' };
    // Set the popup list height
    public height: string = '200px';
    // set the placeholder to the MultiSelect input
    public placeholder: string = 'Select vegetables';
    // set value of enableGroupCheckBox to true
    enableGroupCheckBox: boolean = true;
    // set value of allowFiltering to true
    allowFiltering: boolean = true;
    // set the placeholder to the filterbar
    filterBarPlaceholder: string = "Search Vegetables";
    // set the value of showSelectAll as true
    showSelectAll: boolean = true;
    ngOnInit(): void {
        // set the type of mode for checkbox to visualized the checkbox added
        in li element.
        this.mode = 'CheckBox';
    }
}

```

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 Template support to MultiSelect.](#)

Filtering in Angular Multi select component

The MultiSelect has built-in support to filter data items when `allowFiltering` is enabled. The filter operation starts as soon as you start typing characters in the MultiSelect input.

To display filtered items in the popup, filter the required data and return it to the MultiSelect via `updateData` method by using the `filtering` event.

The following sample illustrates how to query the data source and pass the data to the MultiSelect through the `updateData` method in `filtering` event.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'

```



```

import { Component } from '@angular/core';
import { FilteringEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { DataManager, Query } from '@syncfusion/ej2-data';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='searchData' [fields]='fields' [allowFiltering]='true'
[placeholder]='placeholder' (filtering)='onFiltering($event)'></ejs-
multiselect>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public searchData: { [key: string]: Object }[] = [
    { index: "s1", country: "Alaska" }, { index: "s2", country: "California" },
    { index: "s3", country: "Florida" }, { index: "s4", country: "Georgia" }
  ];
  // map the appropriate column
  public fields: Object = { text: "country", value: "index" };
  // set the placeholder to the MultiSelect input
  public placeholder: string = 'Select countries';
  //Bind the filter event
  public onFiltering: EmitType<FilteringEventArgs> = (e:
FilteringEventArgs) => {
    let query = new Query();
    //frame the query based on search string with filter type.
    query = (e.text !== "") ? query.where("country", "startswith", e.text,
true) : query;
    //pass the filter data source, filter query to updateData method.
    e.updateData(this.searchData, query);
  };
}

```

MAIN.TS

```

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

```

Limit the minimum filter character

When filtering the list items, you can set the limit for character count to raise remote request and fetch filtered data on the MultiSelect. This can be done by manual validation within the filter event handler.

In the following example, the remote request does not fetch the search data until the search key contains three characters.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { FilteringEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='searchData' [fields]='fields' [allowFiltering]='true'
[placeholder]='placeholder' [sortOrder]='sorting' [query]='query'
(filtering)='onFiltering($event)'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  //bind the DataManager instance to dataSource property
  public searchData: DataManager = new DataManager({
    url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // map the appropriate column
  public fields: Object = { text: 'ContactName', value: 'CustomerID' };
  //bind the Query instance to query property
  public query: Query = new Query().select(['ContactName',
'CustomerID']).take(7);
  // set the placeholder to the MultiSelect input
  public placeholder: string = 'Select customers';
  //sort the resulted items
  public sorting: string = 'Ascending';
  //Bind the filter event
  public onFiltering: EmitType<FilteringEventArgs> = (e:
FilteringEventArgs) => {
    // load overall data when search key empty.
    if(e.text == '') e.updateData(this.searchData);
    else{
      // restrict the remote request until search key contains 3
characters.
      if (e.text.length < 3) { return; }
      let query: Query = new Query().select(['ContactName',
'CustomerID']);
      query = (e.text !== '') ? query.where('ContactName', 'startswith',
e.text, true) : query;
      e.updateData(this.searchData, query);
    }
  }
}

```

```
};
}
```

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 filter type

While filtering, you can change the filter type to **contains**, **startsWith**, or **endsWith** for string type within the filter event handler.

In the following examples, data filtering is done with **endsWith** type.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { FilteringEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, ODataV4Adaptor } from '@syncfusion/ej2-data';
import { EmitType } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='searchData' [fields]='fields' [allowFiltering]='true'
[placeholder]='placeholder' [sortOrder]='sorting' [query]='query'
(filtering)='onFiltering($event)'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
    //bind the DataManager instance to dataSource property
    public searchData: DataManager = new DataManager({
      url:
        'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
      adaptor: new ODataV4Adaptor,
      crossDomain: true
    });
    // map the appropriate column
    public fields: Object = { text: 'ContactName', value: 'CustomerID' };
    //bind the Query instance to query property
    public query: Query = new Query().select(['ContactName',
    'CustomerID']).take(7);
    // set the placeholder to the MultiSelect input
```

```

    public placeholder: string = 'Select names';
    //sort the resulted items
    public sorting: string = 'Ascending';
    //Bind the filter event
    public onFiltering: EmitType<FilteringEventArgs> = (e:
FilteringEventArgs) => {
        // load overall data when search key empty.
        if(e.text == '') e.updateData(this.searchData);
        else{
            let query: Query = new Query().select(['ContactName',
'CustomerID']);
            // change the type of filtering
            query = (e.text != '') ? query.where('ContactName', 'endswith',
e.text, true) : query;
            e.updateData(this.searchData, query);
        }
    };
}

```

MAIN.TS

```

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

```

Case sensitive filtering

Data items can be filtered either with or without case sensitivity using the DataManager. This can be done by passing the fourth optional parameter of the `where` clause.

The following example shows how to perform case-sensitive filter.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
import { FilteringEventArgs } from '@syncfusion/ej2-angular-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
import { Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [allowFiltering]='true'
[placeholder]='placeholder' [popupHeight]='popupHeight' [sortOrder]='sorting'
(filtering)='onFiltering($event)'></ejs-multiselect>`
})
export class AppComponent {

```

```

    constructor() {
    }
    //bind the DataManager instance to dataSource property
    public sportsData: { [key: string]: Object }[] = [
        { id: 'game1', sports: 'Badminton' },
        { id: 'game2', sports: 'Football' },
        { id: 'game3', sports: 'Tennis' },
        { id: 'game4', sports: 'Golf' },
        { id: 'game5', sports: 'Hockey' }
    ];
    // map the appropriate column
    public fields: Object = { text: 'sports', value: 'id' };
    // set the placeholder to the MultiSelect input
    public placeholder: string = 'Select games';
    //sort the resulted items
    public sorting: string = 'Ascending';
    //set the height of the popup element
    public popupHeight: string = '250px';
    //Bind the filter event
    public onFiltering: EmitType<FilteringEventArgs> = (e:
    FilteringEventArgs) => {
        // load overall data when search key empty.
        if(e.text == '') e.updateData(this.sportsData);
        else{
            let query: Query = new Query().select(['sports', 'id']);
            //enable the case sensitive filtering by passing false to 4th
            parameter.
            query = (e.text !== '') ? query.where('sports', 'startswith',
            e.text, false) : query;
            e.updateData(this.sportsData, query);
        }
    };
}

```

MAIN.TS

```

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

```

Diacritics Filtering

MultiSelect supports diacritics filtering which will ignore the [diacritics](#) and makes it easier to filter the results in international characters lists when the [ignoreAccent](#) is enabled.

In the following sample,data with diacritics are bound as dataSource for MultiSelect.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({

```

```

imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
],
standalone: true,
selector: 'app-root',
// specifies the template string for the DropDownList component with
change event
template: `<ejs-multiselect id='diacritics' [dataSource]='data'
[allowFiltering]='true' [ignoreAccent]='true' placeholder='e.g. aero'>
    </ejs-multiselect>`
))
export class AppComponent {
    constructor() {
    }
    // create local data
    public data: string[] = [
        'Aeróbics',
        'Aeróbics en Agua',
        'Aerografía',
        'Aeromodelaje',
        'Águilas',
        'Ajedrez',
        'Ala Delta',
        'Álbumes de Música',
        'Alusivos',
        'Análisis de Escritura a Mano'];
}

```

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 bind the data](#)
- [How to group the data using header](#)
- [How to add custom value to the MultiSelect](#)

Custom value in Angular Multi select component

The MultiSelect allows user to add a new non-present option to the component value when [allowCustomValue](#) is enabled. while selecting the new custom value [customValueSelection](#) event will be triggered.

The following sample demonstrates configuration of custom value support with the MultiSelect component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'

```

```

import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [allowCustomValue]='true'
[placeholder]='placeholder'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public sportsData: { [key: string]: Object }[] = [
    { id: 'game1', sports: 'Badminton' },
    { id: 'game2', sports: 'Football' },
    { id: 'game3', sports: 'Tennis' }
  ];
  // map the appropriate column
  public fields: Object = { text: 'sports', value: 'id' };
  // set the placeholder to the MultiSelect input
  public placeholder: string = 'Select games';
}

```

MAIN.TS

```

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

```

Virtualization in MultiSelect Dropdown

MultiSelect Dropdown virtualization is a technique used to efficiently render extensive lists of items while minimizing the impact on performance. This method is particularly advantageous when dealing with large datasets because it ensures that only a fixed number of DOM (Document Object Model) elements are created. When scrolling through the list, existing DOM elements are reused to display relevant data instead of generating new elements for each item. This recycling process is managed internally.

During virtual scrolling, the data retrieved from the data source depends on the popup height and the calculation of the list item height. Enabling the [enableVirtualization](#) option in a MultiSelect Dropdown activates this virtualization technique.

When fetching data from the data source, the [actionBegin](#) event is triggered before data retrieval begins. Then, the [actionComplete](#) event is triggered once the data is successfully fetched.

Furthermore, Incremental Search is supported with virtualization in the MultiSelect component. When a key is typed, the focus is moved to the respective element in the open popup state. In the closed popup state, the popup opens, and focus is moved to the respective element in the popup list based on the

typed key. The Incremental Search functionality is well-suited for scenarios involving remote data binding.

Binding local data

The MultiSelect can generate its list items through an array of complex data. For this, the appropriate columns should be mapped to the [fields](#) property. When using virtual scrolling, the list updates based on the scroll offset value, triggering a request to fetch more data from the server.

In the following example, `id` column and `text` column from complex data have been mapped to the `value` field and `text` field, respectively.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
        text: `Item ${i}`,
      };
      this.records.push(item);
    }
  }
  // maps the appropriate column to fields property
  public fields: object = { text: 'text', value: 'id' };
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}
```

MAIN.TS

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

TEMPLATE.HTML


```

<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='false'
[placeholder]='waterMark'>
    </ejs-multiselect>
  </div>
</div>

```

Binding remote data

The MultiSelect supports retrieval of data from remote data services with the help of **DataManager** component. When using remote data, it initially fetches all the data from the server, triggering the **actionBegin** and **actionComplete** events, and then stores the data locally. During virtual scrolling, additional data is retrieved from the locally stored data, triggering the **actionBegin** and **actionComplete** events at that time as well.

The following sample displays the OrderId from the **Orders** Data Service.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, WebApiAdaptor } from '@syncfusion/ej2-data';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  // bind the DataManager instance to dataSource property
  public customerData: DataManager = new DataManager({
    url:
'https://services.syncfusion.com/angular/production/api/Employees',
    adaptor: new WebApiAdaptor,
    crossDomain: true
  });
  // maps the appropriate column to fields property
  public customerField: { [key: string]: string } = { text: 'OrderID',
value: 'OrderID' };
  // set the placeholder to AutoComplete input
  public waterMark: string = 'OrderID ' ;
}

```

MAIN.TS

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

TEMPLATE.HTML

```
<div id="wrapper" style='margin-top: 20px'>
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='customerData' [fields]='customerField' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
      </ejs-multiselect>
    </div>
  </div>
```

Customizing items count in virtualization

When the `enableVirtualization` property is enabled, the `take` property provided by the user within the Query parameter at the initial state or during the `actionBegin` event will be considered. Internally, it calculates the items that fit onto the current page (i.e., probably twice the amount of the popup's height). If the user-provided take value is less than the minimum number of items that fit into the popup, the user-provided take value will not be considered.

The following sample shows the example for Customizing items count in virtualization.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
import { Query } from '@syncfusion/ej2-data';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
```

```

        text: `Item ${i}`,
    };
    this.records.push(item);
}
}
// maps the appropriate column to fields property
public fields: object = { text: 'text', value: 'id' };
public query: Query = new Query().take(20);
public onBegin: any = (e: any) => {
    e.query = new Query().take(25);
};
// set the placeholder to AutoComplete input
public waterMark: string = 'e.g. Item 1';
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style='margin-top: 20px'>
    <div id='content' style="margin: 0px auto; width:300px;">
        <!-- specifies the virtualization for the MultiSelect component-->
        <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px' [query]='query'
(actionBegin)='onBegin($event)' [enableVirtualization]='true'
[allowFiltering]='false' [placeholder]='waterMark'>
            </ejs-multiselect>
        </div>
    </div>
</div>

```

Grouping with virtualization

The MultiSelect component supports grouping with Virtualization. It allows you to organize elements into groups based on different categories. Each item in the list can be classified using the [groupBy](#) field in the data table. After grouping, virtualization works similarly to local data binding, providing a seamless user experience. When the data source is bound to remote data, an initial request is made to retrieve all data for the purpose of grouping. Subsequently, the grouped data works in the same way as local data binding on virtualization.

The following sample shows the example for Grouping with Virtualization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';

```

```

MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i = 1; i <= 150; i++) {
      let id = 'id' + i;
      let text = `Item ${i}`;
      let group = 'Group A';
      // Generate a random number between 1 and 4 to determine the
group
      const randomGroup = Math.floor(Math.random() * 4) + 1;
      switch (randomGroup) {
        case 1:
          group = 'Group A';
          break;
        case 2:
          group = 'Group B';
          break;
        case 3:
          group = 'Group C';
          break;
        case 4:
          group = 'Group D';
          break;
        default:
          break;
      }
      this.records.push({id, text, group});
    }
  }
  // maps the appropriate column to fields property
  public fields: object = { groupBy: 'group', text: 'text', value: 'id' };
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style="margin-top: 20px">
  <div id='content' style="margin: 0px auto; width:300px;">

```

```

        <!-- specifies the virtualization for the MultiSelect component-->
        <ejs-multiselect id='multiselect-virtualization-grouping'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
        </ejs-multiselect>
    </div>
</div>

```

Filtering with virtualization

The MultiSelect component supports Filtering with Virtualization. The MultiSelect includes a built-in feature that enables data filtering when the [allowFiltering](#) option is enabled. In the context of Virtual Scrolling, the filtering process operates in response to the typed characters. Specifically, the MultiSelect sends a request to the server, utilizing the full data source, to achieve filtering. Before initiating the request, an action event is triggered. Upon successful retrieval of data from the server, an action complete event is triggered. The initial data is loaded when the popup is opened. Whether the filter list has a selection or not, the popup closes.

The following sample shows the example for Filtering with Virtualization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
        text: `Item ${i}`,
      };
      this.records.push(item);
    }
  }
  // maps the appropriate column to fields property
  public fields: object = { text: 'text', value: 'id' };
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}

```

MAIN.TS

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

TEMPLATE.HTML

```
<div id="wrapper" style="margin-top: 20px">
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='true'
[placeholder]='waterMark'>
      </ejs-multiselect>
    </div>
  </div>
```

Checkbox with virtualization

The MultiSelect component supports CheckBox selection with Virtualization. The MultiSelect comes with integrated functionality that allows for the selection of multiple values using checkboxes when the [mode](#) property is configured to **CheckBox**. In the context of Virtual Scrolling, the checkbox render with each list element. based on the checkbox selection and unselection, component value property updated with respective values.

The following sample shows the example for checkbox with Virtualization.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll, CheckBoxSelectionService } from '@syncfusion/ej2-angular-dropdowns';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html',
  providers: [CheckBoxSelectionService]
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
```

```

        id: 'id' + i,
        text: `Item ${i}`,
    };
    this.records.push(item);
}
}
// maps the appropriate column to fields property
public fields: object = { text: 'text', value: 'id' };
public mode?: string = 'CheckBox';
// set the placeholder to AutoComplete input
public waterMark: string = 'e.g. Item 1';
ngOnInit(): void {
    // set the type of mode for checkbox to visualized the checkbox added
    // in li element.
    this.mode = 'CheckBox';
}
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style="margin-top: 20px">
    <div id="content" style="margin: 0px auto; width:300px;">
        <!-- specifies the virtualization for the MultiSelect component-->
        <ejs-multiselect id="multiselect-virtualization"
[dataSource]='records' [fields]='fields' popupHeight='200px' [mode]='mode'
[enableVirtualization]='true' [allowFiltering]='false'
[placeholder]='waterMark'>
            </ejs-multiselect>
        </div>
    </div>
</div>

```

Custom value with virtualization

The MultiSelect component supports custom value with Virtualization. When the [allowCustomValue](#) property is enabled, the MultiSelect enables users to include a new option not currently available in the component value. Upon selecting this newly added custom value, the MultiSelect triggers the [customValueSelection](#) event and also custom value will be added to the end of the complete list.

The following sample shows the example for custom value with Virtualization.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';

```

```

import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-
dropdowns';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
        text: `Item ${i}`,
      };
      this.records.push(item);
    }
  }
  // maps the appropriate column to fields property
  public fields: object = { text: 'text', value: 'id' };
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}

```

MAIN.TS

```

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

```

TEMPLATE.HTML

```

<div id="wrapper" style="margin-top: 20px">
  <div id='content' style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id='multiselect-virtualization'
[dataSource]='records' [fields]='fields' popupHeight='200px'
[enableVirtualization]='true' [allowFiltering]='false'
[allowCustomValue]='true' [placeholder]='waterMark'>
      </ejs-multiselect>
    </div>
  </div>

```

Preselect values with virtualization

The MultiSelect component extends its support for preselected values with Virtualization. When binding values from local or remote data to the MultiSelect component, the corresponding data value is fetched from the server and promptly updated within the component. Moreover, when binding a custom value

to the component, the value is updated within the component, and the bound custom value is seamlessly appended to the end of the complete list.

The following sample shows the example for Preselect value with Virtualization.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
import { MultiSelectComponent, VirtualScroll } from '@syncfusion/ej2-angular-dropdowns';
MultiSelectComponent.Inject(VirtualScroll);
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the virtual-scroll url path
  templateUrl: 'virtual-scroll.html'
})
export class AppComponent {
  public records: { [key: string]: Object }[] = [];
  constructor() {
    for (let i: number = 1; i <= 150; i++) {
      const item: { [key: string]: Object } = {
        id: 'id' + i,
        text: `Item ${i}`,
      };
      this.records.push(item);
    }
  }
  // maps the appropriate column to fields property
  public fields: object = { text: 'text', value: 'id' };
  public value = ['id11', 'id22', 'id33'];
  // set the placeholder to AutoComplete input
  public waterMark: string = 'e.g. Item 1';
}
```

MAIN.TS

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

TEMPLATE.HTML

```
<div id="wrapper" style="margin-top: 20px">
  <div id="content" style="margin: 0px auto; width:300px;">
    <!-- specifies the virtualization for the MultiSelect component-->
    <ejs-multiselect id="multiselect-virtualization"
    [dataSource]='records' [value]='value' [fields]='fields' popupHeight='200px'
```

```
[enableVirtualization]='true' [allowFiltering]='false'
[placeholder]='waterMark'>
  </ejs-multiselect>
</div>
</div>
```

Checkbox in Angular Multi select component

The MultiSelect has built-in support to select multiple values through checkbox, when [mode](#) property set as **CheckBox**.

To use checkbox, inject the **CheckBoxSelection** module in the MultiSelect.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { CheckBoxSelectionService, FilteringEventArgs } from
 '@syncfusion/ej2-angular-dropdowns';
import { EmitType } from '@syncfusion/ej2-base';
import { Query } from '@syncfusion/ej2-data';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' (filtering)='onFiltering($event)'
[mode]='mode' [placeholder]='placeholder'></ejs-multiselect>`,
  providers: [CheckBoxSelectionService]
})
export class AppComponent {
  public mode?: string;
  searchData: { [key: string]: Object; }[] | string[] | number[] |
boolean[] | undefined;
  constructor() {
  }
  //set the data to dataSource property
  public sportsData: Object[] = [
    { id: 'Game1', sports: 'Badminton' },
    { id: 'Game2', sports: 'Basketball' },
    { id: 'Game3', sports: 'Cricket' },
    { id: 'Game4', sports: 'Football' },
    { id: 'Game5', sports: 'Golf' }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { text: 'sports', value: 'id' };
  // set placeholder to MultiSelect input element
  public placeholder: string = 'Select games';
  //Bind the filter event
  public onFiltering: EmitType<FilteringEventArgs> = (e:
FilteringEventArgs) => {
```

```

        let query = new Query();
        //frame the query based on search string with filter type.
        query = (e.text != "") ? query.where("country", "startswith", e.text,
true) : query;
        //pass the filter data source, filter query to updateData method.
        e.updateData((this as any).searchData, query);
    };
    ngOnInit(): void {
        // set the type of mode for checkbox to visualized the checkbox added
        in li element.
        this.mode = 'CheckBox';
    }
}

```

MAIN.TS

```

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

```

Select All

The MultiSelect component has in-built support to select the all list items using **Select All** options in the header.

When the [showSelectAll](#) property is set to true, by default Select All text will show. You can customize the name attribute of the Select All option by using [selectAllText](#).

For the unSelect All option, by default unSelect All text will show. You can customize the name attribute of the unSelect All option by using [unSelectAllText](#). You can customize the name attribute of the unSelect All option by using [unSelectAllText](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { CheckBoxSelectionService } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [mode]='mode'
[selectAllText]='selectAllText' showSelectAll=true
[placeholder]='placeholder'></ejs-multiselect>`,
  providers: [CheckBoxSelectionService]
})
export class AppComponent {
  public mode?: string;
}

```

```

public selectAllText: string | any
constructor() {
}
//set the data to dataSource property
public sportsData: Object[] = [
  { id: 'Game1', sports: 'Badminton' },
  { id: 'Game2', sports: 'Basketball' },
  { id: 'Game3', sports: 'Cricket' },
  { id: 'Game4', sports: 'Football' },
  { id: 'Game5', sports: 'Golf' }
];
// maps the appropriate column to fields property
public fields: Object = { text: 'sports', value: 'id' };
// set placeholder to MultiSelect input element
public placeholder: string = 'Select games';
ngOnInit(): void {
  // set the type of mode for checkbox to visualized the checkbox added
  // in li element.
  this.mode = 'CheckBox';
  // set the select all text to MultiSelect checkbox label.
  this.selectAllText= 'Select All';
}
}

```

MAIN.TS

```

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

```

Selection Limit

Defines the limit of the selected items using [maximumSelectionLength](#).

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { CheckBoxSelectionService } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [mode]='mode'
[maximumSelectionLength]='maximumSelectionLength'
[placeholder]='placeholder'></ejs-multiselect>`,
  providers: [CheckBoxSelectionService]
})
export class AppComponent {

```

```

public mode?: string;
public maximumSelectionLength?: number;
constructor() {
}
//set the data to dataSource property
public sportsData: Object[] = [
    { id: 'Game1', sports: 'Badminton' },
    { id: 'Game2', sports: 'Basketball' },
    { id: 'Game3', sports: 'Cricket' },
    { id: 'Game4', sports: 'Football' },
    { id: 'Game5', sports: 'Golf' }
];
// maps the appropriate column to fields property
public fields: Object = { text: 'sports', value: 'id' };
// set placeholder to MultiSelect input element
public placeholder: string = 'Select games';
ngOnInit(): void {
    // set the type of mode for checkbox to visualized the checkbox added
    // in li element.
    this.mode = 'CheckBox';
    // Sets limitation to the value selection
    this.maximumSelectionLength = 3;
}
}

```

MAIN.TS

```

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

```

Selection Reordering

Using [enableSelectionOrder](#) to Reorder the selected items in popup visibility state.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, OnInit } from '@angular/core';
import { CheckBoxSelectionService } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='sportsData' [fields]='fields' [mode]='mode'
[enableSelectionOrder]='false' [placeholder]='placeholder'></ejs-
multiselect>`,
  providers: [CheckBoxSelectionService]
})

```

```

export class AppComponent {
  public mode?: string;
  constructor() {
  }
  //set the data to dataSource property
  public sportsData: Object[] = [
    { id: 'Game1', sports: 'Badminton' },
    { id: 'Game2', sports: 'Basketball' },
    { id: 'Game3', sports: 'Cricket' },
    { id: 'Game4', sports: 'Football' },
    { id: 'Game5', sports: 'Golf' }
  ];
  // maps the appropriate column to fields property
  public fields: Object = { text: 'sports', value: 'id' };
  // set placeholder to MultiSelect input element
  public placeholder: string = 'Select games';
  ngOnInit(): void {
    // set the type of mode for checkbox to visualized the checkbox added
    // in li element.
    this.mode = 'CheckBox';
  }
}

```

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 bind the data](#)
- [How to filter the bound data](#)
- [How to add custom value to the MultiSelect](#)
- [How to render grouping with checkbox.](#)

Chip customization in Angular Multi select component

The MultiSelect allows the user to customize the selected chip element through the [tagging](#) event. In that event, you can set the custom classes to chip element via that event argument of `setClass` method.

The following sample demonstrates chip-customization with the MultiSelect component.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { Component } from '@angular/core';
import { TaggingEventArgs } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ]
})

```

```

    ],
    standalone: true,
    selector: 'app-root',
    template: `<ejs-multiselect id='multiselectelement' [value]='colorValues'
[dataSource]='colorsData' (tagging)='onTagging($event)' [mode]='box'
[placeholder]='waterMark' [fields]='fields'></ejs-multiselect>`,
  })
  export class AppComponent {
    // define the JSON of data
    public colorsData: { [key: string]: Object }[] = [
      { Color: 'Chocolate', Code: '#75523C' },
      { Color: 'CadetBlue', Code: '#3B8289' },
      { Color: 'DarkOrange', Code: '#FF843D' },
      { Color: 'DarkRed', Code: '#CA3832' },
      { Color: 'Fuchsia', Code: '#D44FA3' },
      { Color: 'HotPink', Code: '#F23F82' },
      { Color: 'Indigo', Code: '#2F5D81' },
      { Color: 'LimeGreen', Code: '#4CD242' },
      { Color: 'OrangeRed', Code: '#FE2A00' },
      { Color: 'Tomato', Code: '#FF745C' }
    ];
    // map the appropriate columns to fields property
    public fields: { [key: string]: string } = { text: 'Color', value: 'Code' };
    // set the value to MultiSelect
    public colorValues: [number | string] | any = ['#75523C', '#4CD242', '#FF745C'];
    // set the placeholder to MultiSelect input element
    public waterMark: string = 'Favorite Colors';
    // set the type of mode for how to visualized the selected items in input element.
    public box: string = 'Box';
    // bind the tagging event
    public onTagging = (e: TaggingEventArgs) => {
      // set the current selected item text as class to chip element.
      e.setClass((e.itemData as any)[this.fields['text']].toLowerCase());
    }
  }
}

```

MAIN.TS

```

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

```

Localization in Angular Multi select component

The Localization library allows you to localize static text content of the

[noRecordsTemplate](#) and [actionFailureTemplate](#) properties according to the culture currently assigned to the MultiSelect.

| Locale key | en-US (default) |

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

noRecordsTemplate	No records found
actionFailureTemplate	The request failed
overflowCountTemplate	+\${count} more..
actionFailureTemplate	\${count} selected

Loading translations

To load translation object to your application, use load function of the **L10n** class.

In the following sample, French culture is set to the MultiSelect and no data is loaded.

Hence, the **noRecordsTemplate** property displays its text in French culture initially,

and if the sample is run offline, the **actionFailureTemplate** property displays its text

appropriately. The **overflowCountTemplate** displays its overflow of the value from MultiSelect and the **totalCountTemplate** displays its total count of the selected value.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit } from '@angular/core';
import { DataManager, ODataV4Adaptor, Query } from '@syncfusion/ej2-data';
// import L10n class for load function
import { L10n, setCulture } from '@syncfusion/ej2-base';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement'
[dataSource]='customerData' [query]='query' [fields]='fields'
[placeholder]='text' [locale]='locale'></ejs-multiselect>`
})
export class AppComponent implements OnInit {
  constructor() {
  }
  //set the placeholder text in french to MultiSelect input
  public text: string = "Sélectionnez un élément";
  // bind remotedata to showcase actionFailureTemplate in offline
  public customerData: DataManager = new DataManager({
    url:
'https://services.odata.org/V4/Northwind/Northwind.svc/Customers',
    adaptor: new ODataV4Adaptor,
    crossDomain: true
  });
  // map appropriate column
  public fields: Object = { text: 'ContactName', value: 'CustomerID' };
  // take 0 item to showcase noRecordsTemplate property
  public query: Query = new Query().select(['ContactName',
'CustomerID']).take(0);
```



```
//set culture to MultiSelect component
public locale: string = 'fr-BE';
ngOnInit(): void {
    L10n.load({
        'fr-BE': {
            'multi-select': {
                'noRecordsTemplate': "Aucun enregistrement trouvé",
                'actionFailureTemplate': "Modèle d'échec d'action",
                'overflowCountTemplate': "+${count} plus..",
                'totalCountTemplate': "${count} choisi"
            }
        }
    });
}
```

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](#)
- [How to bind the data to the combobox](#)

Style in Angular Multi select component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the background color of wrapper element

Use the following CSS to customize the background color of wrapper element.

```
`css
.e-multiselect.e-input-group .e-multi-select-wrapper {
background-color: red;
}
```

Customizing the appearance of the delimiter wrapper element

Use the following CSS to customize the appearance of delimiter wrapper element.

```
`css
.e-multiselect .e-delim-values {
-webkit-text-fill-color: blue;
font-size: 16px;
```

```
font-family: cursive;
}
```

Customizing the appearance of chips

Use the following CSS to customize the appearance of selected chips.

```
`css
.e-multiselect .e-multi-select-wrapper .e-chips .e-chipcontent {
font-family: cursive;
font-size: 20px;
-webkit-text-fill-color: blue;
}
.e-multi-select-wrapper .e-chips {
background-color: aqua;
height: 26px;
}
```

Customizing the dropdown icon's color

Use the following CSS to customize the dropdown icon's color.

```
`css
.e-multiselect.e-input-group .e-input-group-icon, .e-multiselect.e-input-group.e-control-wrapper .e-
input-group-icon:hover {
color: red;
font-size: 14px;
}
```

Customizing the focus color

Use the following CSS to customize the focusing color of input element.

```
`css
.e-multiselect.e-input-group.e-control-wrapper.e-input-focus::before, .e-multiselect.e-input-group.e-
control-wrapper.e-input-focus::after {
background: #c000ff;
}
```

Customizing the disabled component's text color

Use the following CSS to customize the text color when the component is disabled.

```
`css
.e-multiselect.e-disabled .e-multi-select-wrapper .e-delim-values {
-webkit-text-fill-color: red;
}
`
```

Customizing the color of the placeholder text

Use the following CSS to customize the text color of placeholder.

```
`css
.e-multiselect input.e-dropdownbase::placeholder {
color: red;
}
`
```

Customizing the placeholder to add mandatory indicator(*)

Use the following CSS to add the mandatory indicator * to the float label element.

```
`css
.e-input-group.e-control-wrapper.e-float-input .e-float-text::after {
content: "*";
color: red;
}
`
```

Customizing the float label element's focusing color

Use the following CSS to customize the focusing color of float label element.

```
`css
.e-float-input.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-control-
wrapper.e-input-group:not(.e-float-icon-left) .e-float-line::before,.e-float-input.e-input-group:not(.e-
float-icon-left) .e-float-line::after,.e-float-input.e-control-wrapper.e-input-group:not(.e-float-icon-left)
.e-float-line::after {
background-color: #2319b8;
}

.e-multiselect.e-input-group.e-control-wrapper.e-float-input.e-input-focus .e-float-text.e-label-top, .e-
float-input.e-control-wrapper:not(.e-error).e-input-focus input ~ label.e-float-text {
color: #2319b8;
}
`
```

Customizing the outline theme's focus color

Use the following CSS to customize the focusing color of outline theme.

```
`css
.e-outline.e-input-group.e-input-focus:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus.e-control-wrapper:hover:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled):not(.e-float-icon-left),.e-outline.e-input-group.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled),.e-outline.e-input-group.e-control-wrapper.e-input-focus:not(.e-success):not(.e-warning):not(.e-error):not(.e-disabled) {
border-color: #b1bd15;
box-shadow: inset 1px 1px #b1bd15, inset -1px 0 #b1bd15, inset 0 -1px #b1bd15;
}
`
```

Customizing the background color of focus, hover, and active item's

Use the following CSS to customize the background color of focus, hover and active item's.

```
`css
.e-dropdownbase .e-list-item.e-item-focus, .e-dropdownbase .e-list-item.e-active, .e-dropdownbase .e-list-item.e-active.e-hover, .e-dropdownbase .e-list-item.e-hover {
background-color: #1f9c99;
color: #2319b8;
}
`
```

Customizing the appearance of pop-up element

Use the following CSS to customize the appearance of popup element.

```
`css
.e-dropdownbase .e-list-item, .e-dropdownbase .e-list-item.e-item-focus {
background-color: #29c2b8;
color: #207cd9;
font-family: emoji;
min-height: 29px
}
`
```

Customizing the color of the checkbox

Use the following CSS to customize the color of checkbox.

```
`css
.e-popup .e-checkbox-wrapper .e-frame.e-check, .e-popup .e-checkbox-wrapper:hover .e-frame.e-check
{
```

```
background-color: green;
color: white;
}
`
```

Accessibility in Angular Multi select component

The MultiSelect component has been designed, keeping in mind the WAI-ARIA specifications, and applies the WAI-ARIA roles, states, and properties along with **keyboard support**. This component is characterized by complete keyboard interaction support and ARIA accessibility support that makes it easy for people who use assistive technologies (AT) or those who completely rely on keyboard navigation.

The MultiSelect 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 MultiSelect 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 MultiSelect component uses the **Listbox** role, and each list item has an **option** role. The following ARIA attributes denote the MultiSelect state.

ARIA attributes denote the MultiSelect state.

| Properties | Functionalities |

| --- | --- |

| aria-haspopup | Indicates whether the MultiSelect input element has a popup list or not. |

| aria-expanded | Indicates whether the popup list has expanded or not. |

| aria-selected | Indicates the selected option. |

| aria-readonly | Indicates the readonly state of the MultiSelect element. |

| aria-disabled | Indicates whether the MultiSelect component is in a disabled state or not. |

| aria-activedescendent | This attribute holds the ID of the active list item to focus its descendant child element. |

| aria-owns | This attribute contains the ID of the popup list to indicate popup as a child element. |

Keyboard interaction

You can use the following key shortcuts to access the MultiSelect without interruptions.

| Keyboard shortcuts | Actions |

| --- | --- |

| Arrow Down | Set focus at the first item in the MultiSelect when no item selected. Otherwise, moves focus next to the currently selected item. |

| Arrow Up | Moves focus previous to the currently selected one. |

| Page Down | Scrolls down to the next page and set focus to the first item when popup list opens. |

| Page Up | Scrolls up to the previous page and set focus to the first item when popup list opens. |

| Enter | Selects the focused item, and popup list closes when it is in open state. |

| Tab | Focuses on the next TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Shift + tab | Focuses on the previous TabIndex element on the page when the popup is closed. Otherwise, closes the popup list and remains the focus of the component. |

| Alt + Down | Opens the popup list. |

| Alt + Up | Closes the popup list. |

| Esc(Escape) | Closes the popup list when it is in an open state and the currently selected item remains the same. |

| Home | set focus to the first item. |

| End | set focus to the last item. |

In the below sample, focus the MultiSelect component using alt+t keys.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, HostListener, ViewChild } from '@angular/core';
import { MultiSelectComponent } from '@syncfusion/ej2-angular-dropdowns';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement' #samples
[dataSource]='gameList' [fields]='fields'[placeholder]='placeholder'
[popupHeight]='height'></ejs-multiselect>`
})
export class AppComponent {
  @ViewChild('samples')
  public sports?: MultiSelectComponent;
  constructor() {
    //set the data to dataSource property
    public gameList: Object[] = [
      { id: 'Game1', game: 'Badminton' },
      { id: 'Game2', game: 'Basketball' },
      { id: 'Game3', game: 'Cricket' },
      { id: 'Game4', game: 'Football' },
      { id: 'Game5', game: 'Golf' }
    ];
    // maps the appropriate column to fields property
    public fields: Object = { text: 'game', value: 'id' };
    // set the popup list height
    public height: string = '200px';
    // set placeholder to MultiSelect input element
    public placeholder: string = 'Select games';
    @HostListener('document:keyup', ['$event'])
    handleKeyboardEvent(event: KeyboardEvent) {
      if (event.altKey && event.keyCode === 84 /* t */) {
```

```

        // press alt+t to focus the control.
        this.sports!.focusIn();
    }
}
}

```

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 MultiSelect component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the MultiSelect component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the MultiSelect component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Form support in Angular Multi select component

The MultiSelect supports both the reactive and template-driven form-building technologies.

Template-Driven Forms

The template-driven forms uses the `ng` directives in view to handle the forms controls.

To enable the template-driven, import the FormsModule into corresponding app component.

For more details about template-driven Forms refer to: <https://angular.io/guide/forms#template-driven-forms>.

Mention the `name` attribute to MultiSelect element which will be used to identify the

form element. To register an MultiSelect element to ngForm, give the ngModel to it

so the FormsModule will automatically detect the MultiSelect as a form element. After that, the MultiSelect value will be selected based on the ngModel value.

The following example demonstrates how to achieve a two-way data binding.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns';
import { ButtonModule } from '@syncfusion/ej2-angular-buttons';
import { Component, OnInit } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';

```



```

@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'form-support.html'
})
export class AppComponent {
  // defined the array of data
  public skillset: string[] = [
    'ASP.NET', 'ActionScript', 'Basic',
    'C++', 'C#', 'dBase', 'Delphi',
    'ESPOL', 'F#', 'FoxPro', 'Java',
    'J#', 'Lisp', 'Logo', 'PHP'
  ];
  public placeholder: String = 'e.g: ActionScript';
  constructor() {
    skillForm = {
      skillname: null,
      sname: '',
      smail: ''
    };
  }
}

```

MAIN.TS

```

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

```

Reactive Forms

The reactive forms uses the reactive model-driven technique to handle form data between component and view, due to that we also call it as the **model-driven** forms. It's listen the form data changes between App component and view also returns the valid states and values of form elements.

For more details about Reactive Forms refer: <https://angular.io/guide/reactive-forms>.

For the reactive forms you should import a ReactiveFormsModule into app module as well as the FormGroup, FormControl should be imported to app component. The FormGroup is used to declare **formGroupName** for the form and the FormControl is used to declare **formControlName** for form controls.

You can declare the formControlName to MultiSelect as usual. then, you must create a value object to the FormGroup and each value will be the default value of the form control.

The following example demonstrates how to use the reactive forms.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'

```

```

import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, OnInit, Inject } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { BrowserModule } from '@angular/platform-browser';
@Component({
  imports: [
    FormsModule, ReactiveFormsModule, MultiSelectModule, ButtonModule
  ],
  standalone: true,
  selector: 'app-root',
  templateUrl: 'form-support.html'
})
export class AppComponent {
  // defined the array of data
  public skillset: string[] = [
    'ASP.NET', 'ActionScript', 'Basic',
    'C++', 'C#', 'dBase', 'Delphi',
    'ESPOL', 'F#', 'FoxPro', 'Java',
    'J#', 'Lisp', 'Logo', 'PHP'
  ];
  public placeholder: String = 'e.g: ActionScript';
  skillForm: FormGroup | any;
  fb: FormBuilder;
  constructor(@Inject(FormBuilder) private builder: FormBuilder) {
    this.fb = builder;
    this.createForm();
  }
  createForm() {
    this.skillForm = this.fb.group({
      skillname: ['', Validators.required],
      sname: ['', Validators.required],
      smail: ['', Validators.required]
    });
  }
}

```

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

Icons support in Angular Multi select component

You can render **icons** to the list items by mapping the [iconCss](#) fields. This [iconCss](#) fields create a span in the list item with mapped class name to allow styling as per your need. [iconCss](#) fields create a span in the list item with mapped class name to allow styling as per your need.

In the following sample, icon classes are mapped with [iconCss](#) field.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'

```

```
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component } from '@angular/core';
@Component({
  imports: [
    FormsModule, MultiSelectModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the MultiSelect component
  template: `<ejs-multiselect id='multiselectelement' [dataSource]='data'
[fields]='fields' [placeholder]='text'></ejs-multiselect>`
})
export class AppComponent {
  constructor() {
  }
  // defined the array of data
  public data: { [key: string]: Object }[] = [
    { class: 'asc-sort', type: 'Sort A to Z', id: '1' },
    { class: 'dsc-sort', type: 'Sort Z to A ', id: '2' },
    { class: 'filter', type: 'Filter', id: '3' },
    { class: 'clear', type: 'Clear', id: '4' }];
  // map the icon column to iconCSS field.
  public fields: Object = { text: 'type', iconCss: 'class', value: 'id' };
  //set the placeholder to MultiSelect input
  public text: string = 'Select a format';
}
```

MAIN.TS

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

Configure the cascading multi select in Angular Multi select component

The cascading MultiSelect is a series of MultiSelect, where the value of one MultiSelect depends upon another's value. This can be configured by using the `change` event of the parent MultiSelect. Within that change event handler, data has to be loaded to the child MultiSelect based on the selected value of the parent MultiSelect.

The following example, shows the cascade behavior of country, state, and city MultiSelect. Here, the `dataBind` method is used to reflect the property changes immediately to the MultiSelect.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { MultiSelectModule } from '@syncfusion/ej2-angular-dropdowns'
import { Component, ViewChild } from '@angular/core';
import { MultiSelectComponent } from '@syncfusion/ej2-angular-dropdowns';
import { Query, DataManager, Predicate } from '@syncfusion/ej2-data';
@Component({
```

```

imports: [
    FormsModule, MultiSelectModule
],
standalone: true,
selector: 'app-root',
// specifies the template path for MultiSelect component
templateUrl: `cascading.html`
})
export class AppComponent {
    constructor() {
    }
    //define the country MultiSelect data
    public countryData: { [key: string]: Object }[] = [
        { countryName: 'Australia', countryId: '2' },
        { countryName: 'United States', countryId: '1' }
    ];
    //define the state MultiSelect data
    public stateData: { [key: string]: Object }[] = [
        { stateName: 'New York', countryId: '1', stateId: '101' },
        { stateName: 'Virginia ', countryId: '1', stateId: '102' },
        { stateName: 'Tasmania ', countryId: '2', stateId: '105' }
    ];
    //define the city MultiSelect data
    public cityData: { [key: string]: Object }[] = [
        { cityName: 'Albany', stateId: '101', cityId: 201 },
        { cityName: 'Beacon ', stateId: '101', cityId: 202 },
        { cityName: 'Emporia', stateId: '102', cityId: 206 },
        { cityName: 'Hampton ', stateId: '102', cityId: 205 },
        { cityName: 'Hobart', stateId: '105', cityId: 213 },
        { cityName: 'Launceston ', stateId: '105', cityId: 214 }
    ];
    // maps the appropriate column to fields property for country MultiSelect
    public countryFields: Object = { text: 'countryName', value: 'countryId'
};
    // maps the appropriate column to fields property for state MultiSelect
    public stateFields: Object = { text: 'stateName', value: 'stateId' };
    // maps the appropriate column to fields property for city MultiSelect
    public cityFields: Object = { text: 'cityName', value: 'cityId' };
    //set the placeholder to country MultiSelect input
    public countryWatermark: string = "Select countries";
    //set the placeholder to state MultiSelect input
    public stateWatermark: string = "Select states";
    //set the placeholder to city MultiSelect input
    public cityWatermark: string = "Select cities";
    @ViewChild('country')
    public countryObj?: MultiSelectComponent | any;
    @ViewChild('state')
    public stateObj?: MultiSelectComponent | any;
    @ViewChild('city')
    public cityObj?: MultiSelectComponent | any;
    public countryChange(): void {
        //Query the data source based on country MultiSelect selected value
        let pred: Predicate | any;
        if (this.countryObj.value)
            for (var d=0; d<this.countryObj.value.length; d++) {
                if (pred)

```

```

        pred =
pred.or("countryId", 'equal', this.countryObj.value[d]);
        else{
            pred=new
Predicate("countryId", 'equal', this.countryObj.value[d]);
        }
    }
    else{
        this.stateObj.setProperties({enabled:false, values:[]});
        this.cityObj.setProperties({enabled:false, values:[]});
        return;
    }
    // enable the state MultiSelect
    this.stateObj.setProperties({query:new
Query().where(pred), enabled:true, values:[]});
    //clear the existing selection in city MultiSelect
    this.cityObj.setProperties({enabled:false, values:[]});
}
public stateChange(): void {
    //Query the data source based on country MultiSelect selected value
    let pred:Predicate| any,temp:any;
    if(this.stateObj.value)
        for(var d=0;d<this.stateObj.value.length;d++){
            if(pred)
                pred = pred.or("stateId", 'equal', this.stateObj.value[d]);
            else{
                pred=new
Predicate("stateId", 'equal', this.stateObj.value[d]);
            }
        }
    else{
        this.cityObj.setProperties({enabled:false, values:[]});
        return;
    }
    this.cityObj.setProperties({query:new
Query().where(pred), enabled:true, values:[]});
}
}

```

MAIN.TS

```

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

```

CASCADING.HTML

```

<div id="wrapper" style='margin-top: 20px'>
    <div id='content' style="margin: 50px auto 0; width:250px;">
        <br>
        <ejs-multiselect #country id="country"
[dataSource]="countryData" [fields]="countryFields" [changeOnBlur]= "false"
(change)="countryChange()" [placeholder]="countryWatermark"></ejs-
multiselect>
        <div class="padding-top">

```

```

        <ejs-multiselect #state id="state" [dataSource]="stateData"
[fields]="stateFields" [changeOnBlur] = "false" (change)="stateChange()"
[placeholder]="stateWatermark" [enabled]="false"></ejs-multiselect>
    </div>
    <div class="padding-top">
        <ejs-multiselect #city id="city" [dataSource]="cityData"
[fields]="cityFields" [placeholder]="cityWatermark" [enabled]="false"></ejs-
multiselect>
    </div>
</div>
</div>

```

NumericTextBox

Getting started with Angular Numerictextbox component

The following section explains the steps required to create the NumericTextBox component and also it demonstrates the basic usage of the NumericTextBox.

Dependencies

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

```

`javascript
|-- @syncfusion/ej2-angular-inputs
|-- @syncfusion/ej2-angular-base
|-- @syncfusion/ej2-angular-popups
|-- @syncfusion/ej2-angular-buttons
|-- @syncfusion/ej2-angular-splitbuttons
|-- @syncfusion/ej2-inputs
|-- @syncfusion/ej2-base
|-- @syncfusion/ej2-popups
|-- @syncfusion/ej2-buttons
|-- @syncfusion/ej2-splitbuttons
`

```

Setup angular environment

Angular provides the easiest way to set angular CLI projects using [Angular CLI](#) tool.

Install the CLI application globally to your machine.

```

`bash
npm install -g @angular/cli
`

```

Create a new application

```

`bash

```

```
ng new syncfusion-angular-numerictextbox
```

```
,
```

By default, it install the CSS style base application. To setup with SCSS, pass `--style=scss` argument on create project.

Example code snippet.

```
`bash
```

```
ng new syncfusion-angular-numerictextbox --style=scss
```

```
,
```

Navigate to the created project folder.

```
`bash
```

```
cd syncfusion-angular-numerictextbox
```

```
,
```

Installing Syncfusion NumericTextBox Package

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

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

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

Ivy library distribution package

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs --save
```

```
,
```

Angular compatibility compiled package(ngcc)

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

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

```
`bash
```

```
npm install @syncfusion/ej2-angular-inputs@ngcc --save
```

```
,
```

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

```
`bash
```

```
@syncfusion/ej2-angular-inputs:"20.2.38-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 NumericTextBox module

Import NumericTextBox module into Angular application(app.module.ts) from the package

`@syncfusion/ej2-angular-inputs`.

```
`javascript
```

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

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

```
// imports the NumericTextBoxModule for the NumericTextBox component
```

```
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs';
```

```
import { AppComponent } from './app.component';
```

```
@NgModule({
```

```
// declaration of ej2-angular-inputs module into NgModule
```

```
imports: [ BrowserModule , NumericTextBoxModule],
```

```
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-inputs/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-inputs/styles/material.css';
```

```
,
```


Adding NumericTextBox component

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

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

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

@Component({
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  template: <ejs-numerictextbox value='10'></ejs-numerictextbox>
})
export class AppComponent { }
`
```

Note: If you want to refer the combined component styles, please make use of our [CRG](#) (Custom Resource Generator) in your application.

Running the application

After completing the configuration required to render a basic NumericTextBox, run the following command to display the output in your default browser.

```
`ng serve
`
```

The following example illustrates the output in your browser.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  template: `<ejs-numerictextbox value='10'></ejs-numerictextbox>`,
})
export class AppComponent {
  constructor() {
  }
}
```

```
}

```

MAIN.TS

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

```

Range validation

You can set the minimum and maximum range of values in the NumericTextBox using the [min](#) and [max](#) properties, so the numeric value should be in the min and max range.

The validation behavior depends on the [strictMode](#) property.

The below example demonstrates range validation.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets the minimum and maximum range values
  // strictMode has been enabled by default
  template: `
    <ejs-numerictextbox min='10' max='20' value='16' step='2'></ejs-
    numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}

```

MAIN.TS

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

```

Formatting the value

User can set the format of the NumericTextBox component using [format](#) property. The value will be displayed in the specified format, when the component is in focused out state. For more information about formatting the value, refer to this [link](#).

The below example demonstrates format the value by using currency format value `c2`.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [

    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets currency with 2 numbers of decimal places format
  template: `
    <ejs-numerictextbox format='c2' value='10'></ejs-numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

MAIN.TS

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

Precision of numbers

You can restrict the number of decimals to be entered in the NumericTextBox by using the [decimals](#) and [validateDecimalOnType](#) properties.

So, you can't enter the number whose precision is greater than the mentioned decimals.

- If `validateDecimalOnType` is false, number of decimals will not be restricted.

Else, number of decimals will be restricted while typing in the NumericTextBox.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
```

```

import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [

    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // restricts number of decimals to be entered in the NumericTextBox by
  // using validateDecimalOnType property
  // sets number of decimal places to be allowed by the NumericTextBox
  template: `
    <div class='wrap'>
      <ejs-numerictextbox [validateDecimalOnType]='true'
decimals='3' format='n3' value='10' placeholder='ValidateDecimalOnType
Enabled' floatLabelType= 'Auto'></ejs-numerictextbox>
    </div>
    <div class='wrap'>
      <ejs-numerictextbox decimals='3' format='n3' value='10'
placeholder='ValidateDecimalOnType Disabled' floatLabelType= 'Auto'></ejs-
numerictextbox>
    </div>
  `
})
export class AppComponent {
  constructor() {
  }
}

```

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 NumericTextBox, the **value** property supports two-way binding functionality.

The below example demonstrates two-way binding functionality with the NumericTextBox and HTML input element.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [

    NumericTextBoxModule,

```

```

    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component and
  // input element for checking the two-way binding support using value
  property
  template: `
    <div class='e-input-group' style='margin-bottom:30px'>
      <ejs-input class='e-input' type='text' [(ngModel)]='value'
placeholder='Enter a number' />
    </div>
    <ejs-numerictextbox [(value)]='value'></ejs-numerictextbox>
  `
})
export class AppComponent {
  value: any;
  constructor() {
  }
}

```

MAIN.TS

```

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

```

Reactive form

NumericTextBox is a form component and validation is playing vital role in forms to get the valid data.

Here to showcase the NumericTextBox with form validations we have used the reactive form.

For more details about Reactive Forms refer: <https://angular.io/guide/reactive-forms>.

- To use reactive forms, import `ReactiveFormsModule` from the `@angular/forms` package and add it to your `NgModule`'s imports array. In addition to this, `FormGroup`, `FormControl` should be imported to the app component.
- The `FormGroup` is used to declare `formGroupName` for the form. The constructor of this `FormGroup` then takes an object, that can contain sub-form-groups and `FormControls`.
- The `FormControl` is used to declare `formControlName` for form controls.

The following example demonstrates how to use the reactive forms.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule, ReactiveFormsModule } from '@angular/forms'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { ButtonModule } from '@syncfusion/ej2-angular-buttons'
import { Component, Inject } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({

```

```
imports: [
    FormsModule, ReactiveFormsModule, NumericTextBoxModule, ButtonModule
],
standalone: true,
selector: 'app-root',
templateUrl: './template.html',
})
export class AppComponent {
    skillForm?: FormGroup | any;
    build: FormBuilder;
    constructor(@Inject(FormBuilder) private builder: FormBuilder) {
        this.build = builder;
        this.createForm();
    }
    createForm() {
        this.skillForm = this.build.group({
            numeric: ['', Validators.required],
            username: ['', Validators.required],
        });
    }
    get username() { return this.skillForm?.get('username'); }
    get numeric() { return this.skillForm?.get('numeric'); }
    onSubmit() {
        alert("You have entered the value: " + this.numeric?.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

- [How to perform custom validation using FormValidator](#)
- [How to customize the UI appearance of the control](#)
- [How to customize the spin button's up and down arrow](#)
- [How to customize the step value and hide spin buttons](#)
- [How to prevent nullable input in NumericTextBox](#)
- [How to maintain trailing zeros in NumericTextBox](#)

Formats in Angular Numerictextbox component

You can format the value of NumericTextBox using [format](#) property.

The value will be displayed in the specified format when the component is in focused out state. The format string supports both the [standard numeric format string](#) and [custom numeric format string](#) as specified in MSDN.

Standard formats

From the [standard Numeric Formats](#) of MSDN, you can use the numeric related format specifiers such as **n**, **p** and **c** in the NumericTextBox component. By using these format specifiers, you can achieve the percentage and currency textbox behavior also.

The below example demonstrates percentage and currency formats.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [

    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets percentage with 2 numbers of decimal places format
  // sets currency with 2 numbers of decimal places format
  template: `
    <div class='wrap'>
      <ejs-numerictextbox format='p2' value='0.5' min='0' max='1'
step='0.01' placeholder='Percentage format' floatLabelType= 'Auto'></ejs-
numerictextbox>
    </div>
    <div class='wrap'>
      <ejs-numerictextbox format='c2' value='10'
placeholder='Currency format' floatLabelType= 'Auto'></ejs-numerictextbox>
    </div>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

MAIN.TS

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

Custom formats

From the [custom numeric format string](#) of MSDN, you can provide any custom format by combining one or more custom specifiers.

The below examples demonstrate format the value by using currency format string **#** and **0**.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [

    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets the format using custom format string `#`
  // sets the format using custom format string `0`
  template: `
    <div class='wrap'>
      <ejs-numerictextbox format='###.##' value='10'
placeholder='Custom format string #' floatLabelType= 'Auto'></ejs-
numerictextbox>
    </div>
    <div class='wrap'>
      <ejs-numerictextbox format='000.00' value='10'
placeholder='Custom format string 0' floatLabelType= 'Auto'></ejs-
numerictextbox>
    </div>
  `
})
export class AppComponent {
  constructor() {
  }
}

```

MAIN.TS

```

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

```

Globalization in Angular Numerictextbox component

Localization

Localization library allows users to localize the default text contents of the NumericTextBox to different cultures using the [locale](#) property.

In NumericTextBox, spin buttons title for the tooltip will be localized based on the culture.

| Locale key | en-US (default) |

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

| incrementTitle | Increment value |

| decrementTitle | Decrement value |

Loading translations

To load translation object in your application use `load` function of `L10n` class.

The below example demonstrates the NumericTextBox in `German` culture with the spin buttons tooltip.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component, OnInit } from '@angular/core';
import { L10n } from '@syncfusion/ej2-base';
@Component({
  imports: [
    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  // sets `German` culture using the culture value 'de'
  template: `<ejs-numerictextbox locale='de' value='10'></ejs-
numerictextbox>`
})
export class AppComponent implements OnInit {
  constructor() {
  }
  ngOnInit(): void {
    // Load `German` culture to override spin buttons tooltip text
    L10n.load({
      'de': {
        'numerictextbox': {
          incrementTitle: 'Wert erhöhen', decrementTitle:
'Dekrementwert'
        }
      }
    });
  }
}
```

MAIN.TS

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

Internationalization

Internationalization library provides support for formatting and parsing the number by using the official [Unicode CLDR](#) JSON data and also provides the `loadCldr` method to load the culture specific

CLDR JSON data. The NumericTextBox comes with built-in internationalization support to adapt based on culture. For more information about internationalization, refer to this [link](#).

By default, all the Essential JS 2 component are specific to English culture ('en-US').

If you want to go with the different culture other than English, follow the below steps.

- Install the CLDR-Data package by using the below command (it installs the CLDR JSON data). For more information about CLDR-Data, refer to this [link](#).

```
npm install cldr-data --save
```

Once the package installed, you can find the culture specific JSON data under the location `/node_modules/cldr-data`.

- Now import the installed CLDR JSON data into the `app.ts` file.
- Now import the required culture from the installed location to `app.ts` file as like the below code snippets.

```
`typescript
declare var require: any;
loadCldr(
  require('cldr-data/main/de/numbers.json'),
  require('cldr-data/main/de/currencies.json'),
  require('cldr-data/supplemental/numberingSystems.json'),
  require('cldr-data/supplemental/currencyData.json')
);
```

- Set the culture by using the [locale](#) property.

The below example demonstrates the NumericTextBox in German culture with the EUR currency format.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit } from '@angular/core';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
// Here we have referred local json files for preview purpose
import * as numberingSystems from './numberingSystems.json';
import * as currencyData from './currencyData.json';
import * as numbers from './numbers.json';
```

```
import * as currencies from './currencies.json';
loadCldr(numberingSystems, currencyData, numbers, currencies);
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the DropDownList component with
  // change event
  // sets `German` culture using the culture value 'de'
  // sets the `EUR` currency format
  template: `<ejs-numerictextbox locale='de' currency='EUR' format='c2'
value='100'></ejs-numerictextbox>`
})
export class AppComponent implements OnInit {
  constructor() {
  }
  ngOnInit(): void {
    // Load `German` culture to override spin buttons tooltip text
    L10n.load({
      'de': {
        'numerictextbox': {
          incrementTitle: 'Wert erhöhen', decrementTitle:
'Dekrementwert'
        }
      }
    });
  }
}
```

MAIN.TS

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

Right to Left(RTL)

RTL provides an option to switch the text direction and layout of the NumericTextBox component from right to left. It improves the user experiences and accessibility for users who use right-to-left languages (Arabic, Farsi, Urdu, etc.). To enable RTL NumericTextBox, set the [enableRtl](#) to true.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, OnInit } from '@angular/core';
import { loadCldr, L10n } from '@syncfusion/ej2-base';
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
```

```

    selector: 'app-root',
    // specifies the template string for the DropDownList component with
    // change event
    // sets `German` culture using the culture value 'de'
    // sets the 'EUR' currency format
    template: `<ejs-numerictextbox locale='ar-AE' placeholder='أدخل القيمة'
floatLabelType='Auto' enableRtl='true' value='100'></ejs-numerictextbox>`
  })
  export class AppComponent implements OnInit {
    constructor() {
    }
    ngOnInit(): void {
      // Load `German` culture to override spin buttons tooltip text
      L10n.load({
        'ar-AE': {
          'numerictextbox': {
            incrementTitle: 'قيمة الزيادة', decrementTitle: 'قيمة
تناقص'
          }
        }
      });
    }
  }
}

```

MAIN.TS

```

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

```

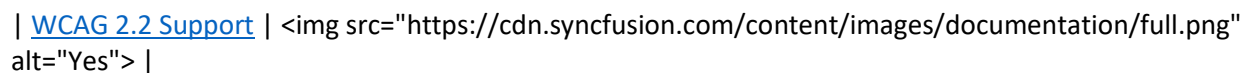
Accessibility in Angular Numerictextbox component

The Numerictextbox 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 Numerictextbox component is outlined below.

| Accessibility Criteria | Compatibility |

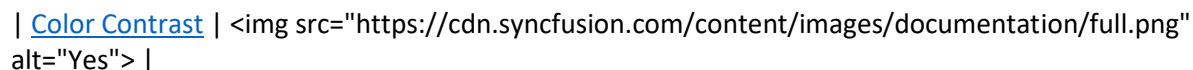
| -- | -- |

| [WCAG 2.2 Support](#) |  alt="Yes"> |

| [Section 508 Support](#) |  alt="Yes"> |

| [Screen Reader Support](#) |  alt="Yes"> |

| [Right-To-Left Support](#) |  alt="Yes"> |

| [Color Contrast](#) |  alt="Yes"> |

```

| Mobile Device Support |  |
| 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 NumericTextBox characterized with complete ARIA Accessibility support which helps to accessible by on-screen readers and other assistive technology devices. This component designed with the reference of the guidelines document given in [WAI ARAI Accessibility practices](#).

The NumericTextBox uses the `spinbutton` role and following ARIA properties to its element based on its state.

| Property | Functionality |

| --- | --- |

| aria-live | The `aria-live` attribute indicates the priority of updates to a live region |

| aria-valuemin | The `aria-valuemin` property specifies the minimum allowable range of the NumericTextBox. |

| aria-valuemax | The `aria-valuemax` property specifies the maximum allowable range of the NumericTextBox. |

| aria-disabled | The `aria-disabled` property indicates disabled state of the NumericTextBox. |

| aria-readonly | The `aria-readonly` property indicates the read-only state of the NumericTextBox. |

| aria-valuenow | The `aria-valuenow` property specifies the current value of the NumericTextBox. |

| aria-invalid | The `aria-invalid` property indicates that the user input is incorrect or not within acceptable ranges. |

| aria-label | The `aria-label` property indicates a string value that labels the NumericTextBox. |

Keyboard interaction

Keyboard interaction of the NumericTextBox component has been designed based on [WAI-ARIA Practices](#) described for the NumericTextBox and it is an alternative to mouse actions to interact with the NumericTextBox.

The below table shows shortcut keys and its corresponding usage.

| **Keyboard shortcuts** | **Actions** |

| --- | --- |

| **Arrow Down** | **Increments the value.** |

| **Arrow Up** | **Decrements the value** |

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets value to the NumericTextBox
  template: `
    <ejs-numerictextbox value='10'></ejs-numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

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 NumericTextBox component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the NumericTextBox component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the NumericTextBox component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

Style appearance in Angular Numerictextbox component

The following content provides the exact CSS structure that can be used to modify the control's appearance based on the user preference.

Customizing the appearance of NumericTextBox wrapper element

Use the following CSS to customize the appearance of wrapper element.

```
`css
/ To specify height and font size /
.e-input-group input.e-input, .e-input-group.e-control-wrapper input.e-input, .e-input-group textarea.e-input, .e-input-group.e-control-wrapper textarea.e-input {
height: 40px;
font-size: 20px;
}
`
```

Customizing the NumericTextBox icons

Use the following CSS to customize the Numeric TextBox icons

```
`css
/ To specify font size and background color /
.e-numeric.e-control-wrapper.e-input-group .e-input-group-icon {
font-size: 20px;
background-color: beige;
}
`
```

How To

Customize the ui appearance of the control in Angular Numerictextbox component

You can change the appearance of the NumericTextBox by adding custom `cssClass` to the component and enabling styles. Refer to the following example to change the appearance of the NumericTextBox.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
```

```
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets the custom css for NumericTextBox
  template: `
    <ejs-numerictextbox cssClass='e-style' value='10' placeholder=
    'Enter Value' floatLabelType= 'Always' ></ejs-numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

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 spin buttons up and down arrow in Angular Numerictextbox component

This section explains about how to change/customize spin up and down icons. You can customize spin button icons using `e-spin-up` and `e-spin-down` classes of those buttons.

You can override the default icons of `e-spin-up` and `e-spin-down` classes using the following CSS code snippets.

```
`css
.e-numeric .e-input-group-icon.e-spin-up:before {
  content: "\e823";
  color: rgba(0, 0, 0, 0.54);
}
.e-numeric .e-input-group-icon.e-spin-down:before {
  content: "\e934";
  color: rgba(0, 0, 0, 0.54);
}
```

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component } from '@angular/core';
```



```
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets value to the NumericTextBox
  template: `
    <ejs-numerictextbox value='10'></ejs-numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

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 step value and hide spin buttons in Angular Numerictextbox component

The spin buttons allow you to increase or decrease the value with the predefined [step](#) value. The visibility of spin buttons can be set using the [showSpinButton](#) property.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { FormsModule } from '@angular/forms'
import { Component } from '@angular/core';
@Component({
  imports: [
    NumericTextBoxModule,
    FormsModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  // sets the step value as '2' to increase/decrease the value by '2'
  // sets the showSpinButton value as 'false' to hide the spin buttons
  template: `
    <ejs-numerictextbox step='2' [showSpinButton]='false' min='10'
    max='100' value='16'></ejs-numerictextbox>
  `
})
export class AppComponent {
  constructor() {
  }
}
```

MAIN.TS

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

Maintain trailing zeros in numerictextbox in Angular Numerictextbox component

By default, trailing zeros disappear when the NumericTextBox gets focus. However, you can use the following sample to maintain the trailing zeros while focusing the NumericTextBox.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  template: `
    <ejs-numerictextbox #numeric='' id="numeric" decimals='2'
    format='n2' value='10' placeholder= 'NumericTextBox' floatLabelType=
    'Always' (change)="onChange($event)" (created)="onCreate($event)" ></ejs-
    numerictextbox>
  `
})
export class AppComponent {
  @ViewChild('formElement') numeric?: any;
  public onChange(args: any) {
    var numericObj = this.numeric.ej2_instances[0];
    numericObj.element.value =
    numericObj.formattedValue(numericObj.decimals, +numericObj.element.value);
  }
  public onCreate(args: any): void {
    document.getElementsByClassName('e-
    numerictextbox')[0].addEventListener('focus', () =>{
      var numericObj = this.numeric.ej2_instances[0] as any;
      numericObj.element.value =
      numericObj.formattedValue(numericObj.decimals, +numericObj.element.value);
    });
  }
  constructor() {
  }
}
```

MAIN.TS

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

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

Perform custom validation using form validator in Angular Numerictextbox component

This section explains how to perform custom validation on the NumericTextBox using FormValidator.

The NumericTextBox will be validated when the value changes or the user clicks the submit button.

Validation can be performed by adding custom validation in the rules collection of the FormValidator.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { FormsModule } from '@angular/forms'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'
import { Component, ViewChild } from '@angular/core';
import { NumericTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
import { FormValidator, FormValidatorModel } from '@syncfusion/ej2-inputs';
@Component({
  imports: [
    FormsModule, NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // Initializes the NumericTextBox
  //Renders submit button for validating the NumericTextBox
  template: `
    <form #formElement class="form-horizontal">
      <div class="form-group">
        <div class="col-sm-6">
          <ejs-numerictextbox #numeric="" id="numeric"
[strictMode]='false' min='10' max='100' name="numeric" placeholder=
'NumericTextbox' floatLabelType= 'Always' (created)="onCreate($event)"
(change)= "onChange($event)"></ejs-numerictextbox>
        </div>
        <button type="button" id="submit_btn" (click)="btnClick()"
style="margin-top: 10px">Submit</button>
      </div>
    </form>
  `
})
export class AppComponent {
  @ViewChild('formElement') element: any;
  @ViewChild('numeric') numeric?: NumericTextBoxComponent | any;
  public formObject?: FormValidator;
  ngAfterViewInit() {
    let customFn: (args: { [key: string]: string }) => boolean = (args: {
[key: string]: string }) => {
      if(this.numeric.value>=10 && this.numeric.value<=100) {
        return true;
      }
      else {
        return false;
      }
    };
    // sets required property in the FormValidator rules collection
```

```

    let options: FormValidatorModel = {
        rules: {
            'numericRange': { required: [true, "Number is required"] },
        },
        //to place the error message in custom position
        customPlacement: (inputElement: HTMLElement, errorElement:
HTMLElement) => {
            (inputElement as HTMLElement |
any).parentNode.parentNode.parentNode.appendChild(errorElement);
        }
    };
    this.formObject = new FormValidator(this.element.nativeElement,
options);
    this.formObject.addRules('numericRange', { range: [customFn, "Please
enter a number between 10 to 100"] });
    var proxy = this;
}
// validates NumericTextBox while value changes
public onChange(args: any){
    if (this.numeric.value != null)
        this.formObject?.validate("numericRange");
}
public onCreate(args: any){
    (document.getElementById("numeric") as
HTMLElement).setAttribute("name", "numericRange");
}
public btnClick(): void {
    // validates the NumericTextBox
    this.formObject?.validate("numericRange");
    // checks for incomplete value and alerts the form
    let ele: HTMLInputElement | any =
<HTMLInputElement>document.getElementById('numeric');
    if (ele.value !== "" && ele.value >=10 && ele.value<=100) {
        alert("Submitted");
    }
}
}

```

MAIN.TS

```

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

```

Prevent nullable input in numerictextbox in Angular Numerictextbox component

By default, the value of the NumericTextBox sets to null. In some applications, the value of the NumericTextBox should not be null at any instance. In such cases, following sample can be used to prevent nullable input in NumericTextBox.

APP.COMPONENT.TS

```

import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { NumericTextBoxModule } from '@syncfusion/ej2-angular-inputs'

```

```

import {Component, ViewChild} from '@angular/core';
import { NumericTextBoxComponent } from '@syncfusion/ej2-angular-inputs';
@Component({
  imports: [
    NumericTextBoxModule
  ],
  standalone: true,
  selector: 'app-root',
  // specifies the template string for the NumericTextBox component
  template: `
    <ejs-numerictextbox #numeric="" id="numeric" placeholder=
'NumericTextBox' floatLabelType= 'Always' (created)="onCreate($event)"
(blur)="onBlur($event)" ></ejs-numerictextbox>
  `
})
export class AppComponent {
  @ViewChild('numeric') public numeric?: NumericTextBoxComponent | any;
  public onCreate(args: any) {
    if (this.numeric.value == null) {
      this.numeric.value = 0;
    }
  }
  public onBlur(args: any) {
    if (args.value == null) {
      this.numeric.element.value = 0;
    }
  }
  constructor() {
  }
}

```

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 Numerictextbox component

This article describes the API migration process of NumericTextBox component from Essential JS 1 to Essential JS 2.

Common

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Triggers on creation | **Event create**
 script onCreate() {} |
Event: created
<ejs-numerictextbox value="120" (created)="onCreate()"></ejs-numerictextbox> **script** public onCreate(): void {} |

| Adding custom classes | **Property** *cssClass*
`<input id="numeric" type="text" ej-numerictextbox value="120" cssClass="custom"/>` | **Property:** *cssClass*
`<ej-numerictextbox value="120" cssClass="custom"></ej-numerictextbox>` |

| Triggers when editor is destroyed | **Event** *destroy*
`<input id="numeric" type="text" ej-numerictextbox value="120" (destroy)="onDestroy()"/>`
`<script>onDestroy() {}` | **Event:** *destroyed*
`<ej-numerictextbox value="120" (destroyed)="onDestroy()"></ej-numerictextbox>`
`<script>public onDestroy(): void {}` |

| Destroys textbox | **Method** *destroy*
`<input id="numeric" type="text" ej-numerictextbox value="100"/>`
`var numericObj = $("#numeric").data("ejNumericTextbox");`
`numericObj.destroy();` | **Method:** *destroy*
`<ej-numerictextbox value="100" #numeric></ej-numerictextbox>`
`@ViewChild("numeric") numeric: NumericTextBoxComponent;`
`numeric.destroy();` |

| Control state | **Property** *enabled*
`<input id="numeric" type="text" ej-numerictextbox value="100" enabled=false/>` | **Property:** *enabled*
`<ej-numerictextbox value="100" enabled=false></ej-numerictextbox>` |

| Persistence | **Property** *enablePersistence*
`<input id="numeric" type="text" ej-numerictextbox value="100" enablePersistence=true/>` | **Property:** *enablePersistence*
`<ej-numerictextbox value="100" enablePersistence=true></ej-numerictextbox>` |

| Right To Left | **Property** *enableRTL*
`<input id="numeric" type="text" ej-numerictextbox value="100" enableRTL=true/>` | **Property:** *enableRtl*
`<ej-numerictextbox value="100" enableRtl=true></ej-numerictextbox>` |

| Triggers when editor is focused in | **Event** *focusIn*
`<input id="numeric" type="text" ej-numerictextbox value="20" (focusIn)="onFocusIn()"/>`
`<script>onFocusIn() {}` | **Event:** *focus*
`<ej-numerictextbox value='12345' (focus) = "onFocus()"></ej-numerictextbox>`
`<script>public onFocus(): void {}` |

| Triggers when editor is focused out | **Event** *focusOut*
`<input id="numeric" type="text" ej-numerictextbox value="100" (focusOut)="onFocusOut()"/>`
`<script>onFocusOut() {}` | **Event:** *blur*
`<ej-numerictextbox value='12345' (blur) = "onBlur()"></ej-numerictextbox>`
`<script>public onBlur(): void {}` |

| Sets Height | **Property** *height*
`<input id="numeric" type="text" ej-numerictextbox value="100" height="40px"/>` | **Can be achieved using,**
`<ej-numerictextbox value="100" cssClass="custom"></ej-numerictextbox>`
`<css>.ej-numerictextbox.custom{height: 40px;</>` |

| HTML Attributes | **Property** *htmlAttributes*
`<input id="numeric" type="text" ej-numerictextbox value="100" [htmlAttributes]= "htmlAttr"/>`
`<script>constructor({</>this.htmlAttr = {name: "numerictextbox"};</>}` | **Can be achieved using**
`<ej-numerictextbox #numeric value="1234" name="numerictextbox"></ej-numerictextbox>` |

| Name of editor | **Property** `name`
 | **Can be achieved using**
`#numeric value="1234" name="numerictextbox"></ej-numerictextbox>` |

| Read only | **Property** `readOnly`
 | **Property:** `readonly`
`readonly=true></ej-numerictextbox>` |

| Rounded corners | **Property** `showRoundedCorner`
 | **Can be achieved using**
`<script>
</ej-numerictextbox #numeric value="1234" floatLabelType="Always" cssClass="e-style"></ej-numerictextbox>
</CSS>
.e-control-wrapper.e-numeric.e-input-group.e-style {
border: 2.5px solid;
border-radius: 1rem;
padding-left: 12px;
}
.e-control-wrapper.e-numeric.e-float-input.e-style .e-float-line::before, .e-control-wrapper.e-numeric.e-float-input.e-style .e-float-line::after{
background: none ;
}
.e-control-wrapper.e-numeric.e-input-group.e-style.e-input-focus{
border: solid grey !important;
}` |

| Spin Button | **Property** `showSpinButton`
 | **Property:** `showSpinButton`
`showSpinButton=false></ej-numerictextbox>` |

| Width | **Property** `width`
 | **Property:** `width`
`width="220px"></ej-numerictextbox>` |

| Clear Button | Not Applicable | **Property:** `showClearButton`
 |

Globalization

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Localization culture | **Property** `locale`
 | **Property:** `locale`
`value="80" locale="de-DE"></ej-numerictextbox>` |

Group

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Group digits in editor | **Property** `groupSize`
 | Not Applicable |

| Group Separator | **Property** `groupSeparator`
 | Not Applicable |

Numeric configuration

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Triggers on value change | **Event** *change*
 | **Script**
 onChange() {} | **Event:** *change*
 ej-numerictextbox value="120" (change)="onChange()">/ej-numerictextbox> | **Script**
 public onChange(): void {} |

| Sets digits allowed after decimal point | **Property** *decimalPlaces*
 | **Property:** *decimals*
 ej-numerictextbox value="80" format="n2" decimals="2">/ej-numerictextbox> |

| Decrement value | Not Applicable | **Method:** *decrement*
 ej-numerictextbox #numeric value="100">/ej-numerictextbox> | **@ViewChild("numeric") numeric:**
 NumericTextBoxComponent; | **numeric.decrement();** |

| Disable the textbox | **Method** *disable*
 | **var numericObj =**
 \$("#numeric").data("ejNumericTextbox"); | **numericObj.disable();** | **Can be achieved**
using | **Script**
 ej-numerictextbox #numeric value="1234">/ej-numerictextbox> | **@ViewChild("numeric") numeric:** NumericTextBoxComponent; | **numeric.enabled = false;**
 |

| Enable the textbox | **Method** *enable*
 | **var numericObj =**
 \$("#numeric").data("ejNumericTextbox"); | **numericObj.enable();** | **Can be achieved**
using | **Script**
 ej-numerictextbox #numeric value="1234">/ej-numerictextbox> | **@ViewChild("numeric") numeric:** NumericTextBoxComponent; | **numeric.enabled = true;**
 |

| Gets value of editor | **Method** *getValue*
 | **var numericObj =**
 \$("#numeric").data("ejNumericTextbox"); | **numericObj.getValue();** | **Method:** *getText*
 ej-numerictextbox value="100" #numeric>/ej-numerictextbox> | **@ViewChild("numeric") numeric:** NumericTextBoxComponent; | **numeric.getText();** |

| Increment value | Not Applicable | **Method:** *increment*
 ej-numerictextbox #numeric value="80">/ej-numerictextbox> | **@ViewChild("numeric") numeric:**
 NumericTextBoxComponent; | **numeric.increment();** |

| Step value | **Property** *incrementStep*
 | **Property:** *step*
 ej-numerictextbox value="100" step="2">/ej-numerictextbox> |

| Sets Maximum value | **Property** *maxValue*
 | **Property:** *max*
 ej-numerictextbox value="100" max="200">/ej-numerictextbox> |

| Sets Minimum value | **Property** *minValue*
 | **Property:** *min*
 ej-numerictextbox value="100" min="20">/ej-numerictextbox> |

| Negative pattern for formatting values | **Property** *negativePattern*
`<input id="numeric" type="text" ej-numerictextbox value="-20" negativePattern: "(n)"/>` | Not Applicable |

| Positive pattern for formatting values | **Property** *positivePattern*
`<input id="numeric" type="text" ej-numerictextbox value="20" positivePattern: "n kg"/>` | Not Applicable |

| Specifies value | **Property** *value*
`<input id="numeric" type="text" ej-numerictextbox value="100" />` | **Property:** *value*
`<ejs-numerictextbox value="100"></ejs-numerictextbox>` |

| Displays hint on editor | **Property** *watermarkText*
`<input id="numeric" type="text" ej-numerictextbox value="80" watermarkText="Enter value" />` | **Property:** *placeholder*
`<ejs-numerictextbox value="100" placeholder="Enter value"></ejs-numerictextbox>` |

| Placeholder float type | Not Applicable | **Property:** *floatLabelType*
`<ejs-numerictextbox value="200" placeholder="Enter value" floatLabelType="Auto"></ejs-numerictextbox>` |

Number Formats

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Set Currency symbol | **Property** *currencySymbol*
`<input id="currency" type="text" ej-currencytextbox value="100" currencySymbol="EUR">` | **Property:** *currency*
`<ejs-numerictextbox value="100" format="c2" currency="EUR"></ejs-numerictextbox>` |

| Number Format | Not Applicable | **Property:** *format*
`<ejs-numerictextbox value="200" format="n2"></ejs-numerictextbox>` |

Validation

| Behavior | API in Essential JS 1 | API in Essential JS 2 |

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

| Strict Mode | **Property** *enableStrictMode*
`<input id="numeric" type="text" ej-numerictextbox value="80" enableStrictMode=true />` | **Property:** *strictMode*
`<ejs-numerictextbox value="80" strictMode=true></ejs-numerictextbox>` |

| Validation on typing | **Property** *validateOnType*
`<input id="numeric" type="text" ej-numerictextbox value="80" validateOnType=true />` | **Property:** *validateDecimalOnType*
`<ejs-numerictextbox value="100" validateDecimalOnType=true></ejs-numerictextbox>` |

| Validation Message | **Property** *validationMessage*
`<input id="numeric" type="text" ej-numerictextbox value="100" />`
`<script>constructor() {this.validationRules = {required: true};this.validationMessage = {required: "Required value"};}` | **Can be achieved using Form Validator**
`<form #formElement class="form-horizontal"><ejs-numerictextbox #numeric="" id="numeric" (created)="onCreate($event)" (change)="onChange($event)"></ejs-numerictextbox></form>`
`let options: FormValidatorModel = {rules: { 'numericRange': { required: [true, "Number is required"] }, customPlacement: (inputElement: HTMLElement, errorElement: HTMLElement) => {inputElement.parentNode.parentNode.appendChild(errorElement);}}`
`this.formObject = new FormValidator(this.element.nativeElement, options);` public

```

onCreate(){<br/>document.getElementById("numeric").setAttribute("name",
"numericRange");<br/>}|
| Validation Rules | Property validationRules<br /><br /><input id="numeric" type="text" ej-
numerictextbox value="100" /><br /><constructor() {<br /><br /><script<br />this.validationRules =
{required: true};<br />} | Can be achieved using Form Validator<br/><form #formElement
class="form-horizontal"><br/><ejs-numerictextbox #numeric="" id="numeric"
(created)="onCreate($event)" (change)="onChange($event)"></ejs-
numerictextbox><br/></form><br/>let options: FormValidatorModel = {<br/> rules:
{<br/>'numericRange': { required: [true] },<br/>},<br/>customPlacement: (inputElement:
HTMLElement, errorElement: HTMLElement) =>
{<br/>inputElement.parentNode.parentNode.parentNode.appendChild(errorElement);<br/>}<br
/>};<br/>this.formObject = new FormValidator(this.element.nativeElement, options);<br/>public
onCreate(){<br/>document.getElementById("numeric").setAttribute("name",
"numericRange");<br/>}|

```

Pager

Getting started with Angular Pager component

This section explains you the steps required to create a simple Pager and demonstrate the basic usage of the Pager component in Angular environment.

Setup Angular Environment

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

To install Angular CLI use the following command.

```

`bash
npm install -g @angular/cli
`

```

Create an Angular Application

Start a new Angular application using below Angular CLI command.

```

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

```

Installing Syncfusion Pager 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-grids](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-grids --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-grids@ngcc](#) package to the application.

```
`bash
npm install @syncfusion/ej2-angular-grids@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-grids:"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 Pager Module

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

```
`typescript
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { PagerModule } from '@syncfusion/ej2-angular-grids';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
```

PagerModule

```
],
bootstrap: [AppComponent]
})
export class AppModule { }
```

Adding CSS reference

The 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-angular-grids/styles/material.css';
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
```

Adding Pager component

Modify the template in `[src/app/app.component.ts]` file to render the Angular pager component. Add the Angular pager by using `<ejs-pager>` 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 Pager component
  template: `<ejs-pager [totalRecordsCount]='20'>
</ejs-pager>`
})
export class AppComponent implements OnInit {
  ngOnInit(): void {
  }
}
```

Page Size

`pageSize` value defines the number of records to be displayed per page. The default value for the `pageSize` is 12.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PagerModule } from '@syncfusion/ej2-angular-grids'
```

```
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [

    PagerModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-pager [pageSize]= '1' [totalRecordsCount]='20'>
    </ejs-pager>`
})
export class AppComponent implements OnInit{
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Page Count

pageCount value defines the number of pages to be displayed in the pager component for navigation.

The default value for **pageCount** is 10 and value will be updated based on [totalRecordsCount](#)

and [pageSize](#) values.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PagerModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [

    PagerModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-pager [pageSize]='1' [pageCount]='3'
[totalRecordsCount]='20'>
    </ejs-pager>`
})
export class AppComponent implements OnInit{
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

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

Run the application

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

```
`javascript
ng serve --open
`
```

Output will be appears as follows.

APP.COMPONENT.TS

```
import { NgModule } from '@angular/core'
import { BrowserModule } from '@angular/platform-browser'
import { PagerModule } from '@syncfusion/ej2-angular-grids'
import { Component, OnInit } from '@angular/core';
@Component({
  imports: [

    PagerModule
  ],
  standalone: true,
  selector: 'app-root',
  template: `<ejs-pager [pageSize]='8' [pageCount]='3'
[totalRecordsCount]='20'>
      </ejs-pager>`
})
export class AppComponent implements OnInit{
  ngOnInit(): void {
  }
}
```

MAIN.TS

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

Style and appearance in Angular Pager component

To modify the Pager appearance, you need to override the default CSS of Pager. Please find the CSS structure that can be used to modify the Pager appearance. Also, you have an option to create your own custom theme for all the JavaScript controls using our [Theme Studio](#).

Customizing the Pager

Use the below CSS to customize the Pager root element.

```
`css
```

```
.e-pager {  
background-color: #deecf9;  
}  
`css
```

Customizing the Pager container element

Use the below CSS to customize the pager container element.

```
`css  
.e-pager .e-pagercontainer {  
background-color: #deecf9;  
}  
`css
```

Customizing the Pager navigation elements

Customize the pager navigation elements, using the below selector.

```
`css  
.e-pager .e-prevpagedisabled,  
.e-pager .e-prevpage,  
.e-pager .e-nextpage,  
.e-pager .e-nextpagedisabled,  
.e-pager .e-lastpagedisabled,  
.e-pager .e-lastpage,  
.e-pager .e-firstpage,  
.e-pager .e-firstpagedisabled {  
background-color: #deecf9;  
}  
`css
```

Customizing the Pager page numeric link elements

Use the below CSS to customize the pager current page numeric link elements.

```
`css  
.e-pager .e-numericitem {  
border-radius: initial;  
}  
`css
```

Customizing the Pager current page numeric element

Using this CSS, you can customize the pager current page numeric item.

```
`css
.e-pager .e-currentitem {
background-color: #0078d7;
}
`
```

Accessibility in Angular Pager component

Accessibility is achieved in the Pager component through WAI-ARIA standard and keyboard navigations. The Pager features can be effectively accessed through assistive technologies such as screen readers.

WAI-ARIA

WAI-ARIA (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components.

The following ARIA attributes are used in the Pager:

- pager (data-role)
- aria-selected (attribute)
- aria-owns (attribute)
- aria-label (attribute)

Keyboard navigation

Pager functionalities can be interactive with keyboard shortcuts.

The following keyboard shortcuts are supported by the Pager.

Interaction Keys | Description

Pager | |

Alt + J | Focus on the first pager item.

Tab / Shift + Tab | Focus on the next/previous pager item.

Enter / Space | Select the currently focused page.

Right Arrow / PageDown | Navigate to next page.

Left Arrow / PageUp | Navigate to previous page.

Home / End | Navigate to first and last page.

PDF Viewer

Getting Started

Getting started with Standalone PDF Viewer component

This section explains the steps required to create a simple Standalone Angular PDF Viewer and demonstrates the basic usage of the PDF Viewer control in a Angular CLI application.

Setup Angular Environment

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

To install Angular CLI globally use the following command.

```
`bash
npm install -g @angular/cli@16.0.1
`
```

Note: Use the command **npm install --save @angular/cli@12.0.2** to install the latest Angular CLI version 12.0.2

Create an Angular Application

Start a new Angular application using the Angular CLI command as follows.

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

Installing Syncfusion PDF Viewer package

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

- To install PDF Viewer component, use the following command.

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

- Copy the contents of the ej2-pdfviewer-lib folder from `./node_modules/@syncfusion/ej2-pdfviewer/dist` to the `src/assets` directory using the command:

```
`bash
cp -R ./node_modules/@syncfusion/ej2-pdfviewer/dist/ej2-pdfviewer-lib src/assets/ej2-pdfviewer-lib
`
```

- Confirm that there is an 'ej2-pdfviewer-lib' directory within your public directory, housing the assets of the PDF Viewer library.
- Validate that your server has been configured to utilize the Content-Type: application/wasm MIME type. Additional information can be found in the [Troubleshooting](#) section.

Registering PDF Viewer Module

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

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

```
import { BrowserModule } from '@angular/platform-browser';
// import the PdfViewer Module for the PDF Viewer component
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
import { AppComponent } from './app.component';
@NgModule({
//declaration of ej2-angular-pdfviewer module into NgModule
imports: [BrowserModule, PdfViewerModule],
declarations: [AppComponent],
bootstrap: [AppComponent],
providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService, PageOrganizerService]
})
export class AppModule { }
`
```

Adding CSS reference

Add the Angular PDF Viewer component's styles as given below in `src/styles.css` file.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-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-pdfviewer/styles/material.css';
@import '../node_modules/@syncfusion/ej2-notifications/styles/material.css';
`
```

Adding PDF Viewer component

Add the Angular PDF Viewer by using `<ejs-pdfviewer>` selector in `template` section of the `src/app/app.component.ts` file to render the PDF Viewer component.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService, PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-pdfviewer-lib";
  ngOnInit(): void {
  }
}
```

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 { Component, OnInit } from '@angular/core';
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
}
```

MAIN.TS

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

Limitation over Server-Backed PDF Viewer to Standalone PDF Viewer control

When comparing a Standalone PDF Viewer to a Server-Backed PDF Viewer control, it's crucial to understand the limitations that the Standalone PDF Viewer may have in comparison. These limitations are important to consider

[PNG Image Support](#)

The Standalone PDF Viewer does not have the capability to utilize PNG format for adding images to handwritten annotations ,custom stamp ,signature and initial form fields. It's important to be aware that only certain image formats, such as JPEG, are compatible for these purposes.

[Local File Access](#)

- The Standalone PDF Viewer control does not have the capability to directly access and load local physical files from a user's device. As a result, it is not possible to use a documentPath to load a PDF file directly from a local server within the viewer.
- The Standalone PDF Viewer allows users to export annotations and form fields from the viewer, it's important to be aware that the viewer does not support the direct import of annotations and form fields from a locally specified file path. In other words, you can extract annotations and form fields from the viewer, but you cannot reintroduce them into the viewer from external sources by specifying a file path located on your local device.

Note: These limitations are temporary and are expected to be addressed in the near future.

[View sample in GitHub.](#)

[Getting started with PDF Viewer component](#)

This section explains the steps required to create a simple Angular PDF Viewer and demonstrates the basic usage of the PDF Viewer control in a Angular CLI application.

[Setup Angular Environment](#)

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

To install Angular CLI globally use the following command.

```
`bash
npm install -g @angular/cli@16.0.1
`
```

Note: Use the command **npm install --save @angular/cli@12.0.2** to install the latest Angular CLI version 12.0.2

[Create an Angular Application](#)

Start a new Angular application using the Angular CLI command as follows.

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

[Installing Syncfusion PDF Viewer package](#)

All the available Essential JS 2 packages are published in [npmjs.com](#) registry. To install PDF Viewer component, use the following command.

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

Registering PDF Viewer Module

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

```
`typescript
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
// import the PdfViewer Module for the PDF Viewer component
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
import { AppComponent } from './app.component';
@NgModule({
//declaration of ej2-angular-pdfviewer module into NgModule
imports: [BrowserModule, PdfViewerModule],
declarations: [AppComponent],
bootstrap: [AppComponent],
providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService, PageOrganizerService]
})
export class AppModule { }
```

Adding CSS reference

Add the Angular PDF Viewer component's styles as given below in `src/styles.css` file.

```
`css
@import '../node_modules/@syncfusion/ej2-base/styles/material.css';
@import '../node_modules/@syncfusion/ej2-buttons/styles/material.css';
@import '../node_modules/@syncfusion/ej2-dropdowns/styles/material.css';
@import '../node_modules/@syncfusion/ej2-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-pdfviewer/styles/material.css';
@import '../node_modules/@syncfusion/ej2-notifications/styles/material.css';
`
```

Adding PDF Viewer component

Add the Angular PDF Viewer by using `<ejs-pdfviewer>` selector in `template` section of the `src/app/app.component.ts` file to render the PDF Viewer component.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService, PageOrganizerService]
})
export class AppComponent implements OnInit {
  public service = 'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
  ngOnInit(): void {
```

```

}
}
`

```

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 { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService,
  ToolbarService, NavigationService, AnnotationService, TextSearchService,
TextSelectionService,
  PrintService, FormDesignerService, FormFieldsService, PageOrganizerService }
from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer
      id="pdfViewer"
      [serviceUrl]='service'
      [documentPath]='document'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [LinkAnnotationService, BookmarkViewService,
MagnificationService,ThumbnailViewService, ToolbarService
    , NavigationService, AnnotationService, TextSearchService,
TextSelectionService, PrintService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  ngOnInit(): void {
  }
}

```

MAIN.TS

```

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

```


For PDF Viewer serviceUrl creation, follow the steps provided in the [link](#)

How to run the PDF Viewer web service

1. Download the sample from the [Web service sample in GitHub](#) link.
2. Navigate to the ASP.NET Core folder and open it in the command prompt.
3. Use the below command to restore the required packages.

```
`sh
dotnet restore
`
```

4. Use the below command to run the web service.

```
`sh
dotnet run
`
```

5. You can see that the PDF Viewer server instance runs in the local host with the port number `localhost:5001` and navigate to the PDF Viewer Web control `localhost:5001/pdfviewer` which returns the default get response method. We can bind the link to the `serviceUrl` property of PDF Viewer as below.

```
`javascript
export class AppComponent implements OnInit {
  public service = 'https://localhost:5001/pdfviewer';
  public document = 'PDF_Succinctly.pdf';
  ngOnInit(): void {
  }
`
```

Note: When configuring the server-backed PDF viewer, it's essential to understand that there is no need to include the pdfium.js and pdfium.wasm files. Unlike the standalone PDF viewer, which relies on these files for local rendering, the server-backed PDF viewer fetches and renders PDFs directly from the server. Consequently, you can exclude the copy command for deployment process, as they are not required to load and display PDFs in this context.

Note: For hosting the web service on the Linux platform, ensure to include the [SkiaSharp.NativeAssets.Linux](#). Additionally, for AWS environments, utilize the following packages:

Amazon Web Services (AWS)	NuGet package name
---	---

| AWS Lambda | [SkiaSharp.NativeAssets.Linux](#) |

| AWS Elastic Beanstalk | [SkiaSharp.NativeAssets.Linux.NoDependencies v2.88.6](#) |

[View sample in GitHub.](#)

Feature modules

The [Angular PDF Viewer](#) features are segregated into individual feature-wise modules to enable selectively referencing in the application. The required modules should be injected to extend its functionality. The following are the selective modules of PDF Viewer that can be included as required:

The available PdfViewer modules are:

- **Toolbar**:- Built-in toolbar for better user interaction.
- **Magnification**:- Perform zooming operation for better viewing experience.
- **Navigation**:- Easy navigation across the PDF pages.
- **LinkAnnotation**:- Easy navigation within and outside of the PDF document.
- **ThumbnailView**:- Easy navigation with in the PDF document.
- **BookmarkView**:- Easy navigation based on the bookmark content of the PDF document.
- **TextSelection**:- Select and copy text from a PDF file.
- **TextSearch**:- Search a text easily across the PDF document.
- **Print**:- Print the entire document or a specific page directly from the browser.
- **Annotation**:- Annotations can be added or edited in the PDF document.
- **FormFields**:- Preserve the form fields in the PDF document.
- **FormDesigner**:- Form fields can be added or edited in the PDF document.
- **StickyNotesAnnotation**:- Adding sticky notes to the PDF document.

In addition to injecting the required modules in your application, enable corresponding properties to extend the functionality for a PDF Viewer instance.

Refer to the following table.

Module	Property to enable the functionality for a PDF Viewer instance
---	---
Toolbar	<code><ejs-pdfviewer enableToolbar= true ></ejs-pdfviewer></code>
Magnification	<code><ejs-pdfviewer enableMagnification= true ></ejs-pdfviewer></code>
Navigation	<code><ejs-pdfviewer enableNavigation= true ></ejs-pdfviewer></code>
LinkAnnotation	<code><ejs-pdfviewer enableHyperlink= true ></ejs-pdfviewer></code>
ThumbnailView	<code><ejs-pdfviewer enableThumbnail= true ></ejs-pdfviewer></code>
BookmarkView	<code><ejs-pdfviewer enableBookmark= true ></ejs-pdfviewer></code>
TextSelection	<code><ejs-pdfviewer enableTextSelection= true ></ejs-pdfviewer></code>
TextSearch	<code><ejs-pdfviewer enableTextSearch= true ></ejs-pdfviewer></code>
Print	<code><ejs-pdfviewer enablePrint= true ></ejs-pdfviewer></code>
Annotation	<code><ejs-pdfviewer enableAnnotation= true ></ejs-pdfviewer></code>

```
|FormFields|<ejs-pdfviewer enableFormFields= true ></ejs-pdfviewer>|
|FormDesigner|<ejs-pdfviewer enableFormDesigner= true ></ejs-pdfviewer>|
|StickyNotesAnnotation|<ejs-pdfviewer enableStickyNotesAnnotation= true ></ejs-pdfviewer>|
```

See also

- [Toolbar items](#)
- [Toolbar customization](#)

Open PDF files in Angular PDF Viewer component

You might need to open and view the PDF files from various location. In this section, you can find the information about how to open PDF files from URL, database, local file system, and as base64 string.

Opening a PDF from URL

If you have your PDF files in the web, you can open it in the viewer using URL.

Step 1: Create a Simple PDF Viewer Sample in Angular

Start by following the steps provided in this [link](#) to create a simple PDF viewer sample in Angular. This will give you a basic setup of the PDF viewer component.

Step 2: Modify the PdfViewerController.cs File in the Web Service Project

1. Create a web service project in .NET Core 3.0 or above. You can refer to this [link](#) for instructions on how to create a web service project.
2. Open the PdfViewerController.cs file in your web service project.
3. Modify the Load() method to open it in the viewer using URL

```
`csharp
public IActionResult Load([FromBody] Dictionary<string, string> jsonData)
{
    // Initialize the PDF viewer object with memory cache object
    PdfRenderer pdfviewer = new PdfRenderer(_cache);
    MemoryStream stream = new MemoryStream();
    object jsonResult = new object();
    if (jsonObject != null && jsonObject.ContainsKey("document"))
    {
        if (bool.Parse(jsonObject["isFileName"]))
        {
            string documentPath = GetDocumentPath(jsonData["document"]);
            if (!string.IsNullOrEmpty(documentPath))
            {
```

```

byte[] bytes = System.IO.File.ReadAllBytes(documentPath);
stream = new MemoryStream(bytes);
}
else
{
string fileName = jsonData["document"].Split(new string[] { "://" }, StringSplitOptions.None)[0];
if (fileName == "http" || fileName == "https")
{
WebClient WebClient = new WebClient();
byte[] pdfDoc = WebClient.DownloadData(jsonData["document"]);
stream = new MemoryStream(pdfDoc);
}
else
{
return this.Content(jsonData["document"] + " is not found");
}
}
}
else
{
byte[] bytes = Convert.FromBase64String(jsonObject["document"]);
stream = new MemoryStream(bytes);
}
}
jsonResult = pdfviewer.Load(stream, jsonObject);
return Content(JsonConvert.SerializeObject(jsonResult));
}
,

```

Step 3: Set the PDF Viewer Properties in React PDF viewer component

Modify the `serviceUrl` property of the PDF viewer component with the accurate URL of your web service project, replacing `https://localhost:44396/pdfviewer` with the actual URL of your server. Modify the `documentPath` with the correct PDF Document URL want to load.

`typescript

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

```
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='documentPath'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,ThumbnailViewService,
ToolbarService, NavigationService, AnnotationService, TextSearchService,
TextSelectionService, PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  // Replace the "localhost:44396" with the actual URL of your server
  public service = 'https://localhost:44396/pdfviewer';
  // Replace correct PDF Document URL want to load
  public documentPath="https://cdn.syncfusion.com/content/PDFViewer/flutter-succinctly.pdf"
}
```

[View sample in GitHub](#)

Opening a PDF from base64 data

The following steps explains how the PDF file can be loaded in PDF Viewer as base64 string.

Step 1: Create a Simple PDF Viewer Sample in Angular

Start by following the steps provided in this [link](#) to create a simple PDF viewer sample in Angular. This will give you a basic setup of the PDF viewer component.

Step 2: Use the following code snippet to load the document from Base64 string.

```
`html
<button (click)="load()">LoadDocumentFromBase64</button>
```

```

`typescript
load() {
var viewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
viewer.load(
"data:application/pdf;base64,.....",
null
);
}
`

```

[View sample in GitHub](#)

Saving PDF file

After editing the PDF file with various annotation tools, you will need to save the updated PDF to the server, database, or local file system.

Save PDF file to Server

Need to save the modified PDF back to a server. To achieve this, proceed with the following steps

Step 1: Create a Simple PDF Viewer Sample in Angular

Start by following the steps provided in this [link](#) to create a simple PDF viewer sample in Angular. This will give you a basic setup of the PDF viewer component.

Step 2: Modify the PdfViewerController.cs File in the Web Service Project

1. Create a web service project in .NET Core 3.0 or above. You can refer to this [link](#) for instructions on how to create a web service project.
2. Open the PdfViewerController.cs file in your web service project.
3. Modify the Download() method to open it in the viewer using URL

```

`csharp
public IActionResult Download([FromBody] Dictionary<string, string> jsonObject)
{
//Initialize the PDF Viewer object with memory cache object
PdfRenderer pdfviewer = new PdfRenderer(_cache);
string documentBase = pdfviewer.GetDocumentAsBase64(jsonObject);
MemoryStream stream = new MemoryStream();
string documentName = jsonObject["document"];
string result = Path.GetFileNameWithoutExtension(documentName);
string fileName = result + "_downloaded.pdf";

```

```
// Save the file on the server
string serverFilePath = @"Path to where you need to save your file in the server";
string filePath = Path.Combine(serverFilePath, fileName);
using (FileStream fileStream = new FileStream(filePath, FileMode.Create))
{
    //Saving the new file in root path of application
    stream.CopyTo(fileStream);
    fileStream.Close();
}
return Content(documentBase);
}
```

Step 3: Set the PDF Viewer Properties in React PDF viewer component

Modify the `serviceUrl` property of the PDF viewer component with the accurate URL of your web service project, replacing `https://localhost:44396/pdfviewer` with the actual URL of your server. Modify the `documentPath` with the correct PDF Document URL want to load.

```
`typescript
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
    selector: 'app-container',
    // specifies the template string for the PDF Viewer component
    template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='documentPath'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
    providers: [ LinkAnnotationService, BookmarkViewService, MagnificationService,ThumbnailViewService,
ToolbarService, NavigationService, AnnotationService, TextSearchService,
```

```
TextSelectionService, PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  // Replace the "localhost:44396" with the actual URL of your server
  public service = 'https://localhost:44396/pdfviewer';
  // Replace correct PDF Document URL want to load
  public documentPath="PDF_Succinctly.pdf"
}
`
```

[View sample in GitHub](#)

[Download PDF file as a copy](#)

In the built-in toolbar, you have an option to download the updated PDF to the local file system, you can use it to download the PDF file.

```
`html
<button (click)="downloadClicked()">Download</button>
`

`typescript
downloadClicked() {
  var viewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
  viewer.download();
}
`
```

Built-in toolbar in Angular PDF Viewer component

The PDF Viewer comes with a powerful built-in toolbar to execute important actions such as page navigation, text search, view mode, download, print, bookmark, and thumbnails.

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

Option	Description
OpenOption	This option provides an action to load the PDF documents to the PDF Viewer.
PageNavigationTool	This option provides an action to navigate the pages in the PDF Viewer. It contains GoToFirstPage, GoToLastPage, GotoPage, GoToNext, and GoToLast tools.
MagnificationTool	This option provides an action to magnify the pages either with predefined or user defined zoom factors in the PDF Viewer. Contains ZoomIn, ZoomOut, Zoom, FitPage and FitWidth tools
PanTool	This option provides an action for panning the pages in the PDF Viewer.
SelectionTool	This option provides an action to enable/disable the text selection in the PDF Viewer.

| SearchOption | This option provides an action to search a word in the PDF documents. |

| PrintOption | This option provides an action to print the PDF document being loaded in the PDF Viewer. |

| DownloadOption | This Download option provides an action to download the PDF document that has been loaded in the PDF Viewer. |

| UndoRedoTool | This tool provides options to undo and redo the annotation actions performed in the PDF Viewer. |

| AnnotationEditTool | This tool provides options to enable or disable the edit mode of annotation in the PDF Viewer. |

| CommentTool | This tool facilitates the addition of sticky notes to the pages of PDF documents in the PDF Viewer. |

Show/Hide the default toolbar

The PDF Viewer has an option to show or hide the complete default toolbar. You can achieve this by using following two ways.

- **Show/Hide toolbar using enableToolbar API as in the following code snippet**

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[toolbarSettings]='toolbarSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
```

```
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer id="pdfViewer"
      [serviceUrl]='service'
      [documentPath]='document'
      [toolbarSettings]='toolbarSettings'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    AnnotationService, TextSearchService, TextSelectionService,
    PrintService]
})
export class AppComponent implements OnInit {
  public service =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
}
```

- **Show/Hide toolbar using showToolbar as in the following code snippet**

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.toolbar.showToolbar(false);
}
</script>
`
```

Show/Hide the default toolbaritem

The PDF Viewer has an option to show or hide these grouped items in the default toolbar.

- **Show/Hide toolbaritem using toolbarSettings as in the following code snippet.**

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
```

```

TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[toolbarSettings]='toolbarSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public toolbarSettings = { showTooltip: true, toolbarItems: 'DownloadOption'
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[toolbarSettings]='toolbarSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

```

```
public toolbarSettings = { showTooltip: true, toolbarItems: 'DownloadOption'
}
}
```

- **Show/Hide toolbaritem using showToolbaritem as in the following code snippet**

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.toolbar.showToolbarItem(new Array("DownloadOption"), true);
}
</script>
`
```

Show/Hide the left toolbar with the thumbnails and bookmarks

The PDF Viewer has an option to show or hide the left toolbar with the thumbnails and bookmarks using enableNavigationToolbar API as in the following code sample.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[enableNavigationToolbar]="false"
[toolbarSettings]="toolbarSettings"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public toolbarSettings = { showTooltip: true, toolbarItems: ['OpenOption'] };
}
```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[enableNavigationToolbar]="false"
[toolbarSettings]="toolbarSettings"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public toolbarSettings = { showTooltip: true, toolbarItems: ['OpenOption'] };
}

```

See also

- [Toolbar customization](#)
- [Feature Modules](#)

Navigation in Angular PDF Viewer component

The ASP.NET Core PDF Viewer supports different internal and external navigations.

Toolbar page navigation option

The default toolbar of PDF Viewer contains the following navigation options

- **Go to page:-** Navigates to the specific page of a PDF document.
- **Show next page:-** Navigates to the next page of PDF a document.
- **Show previous page:-** Navigates to the previous page of a PDF document.
- **Show first page:-** Navigates to the first page of a PDF document.
- **Show last page:-** Navigates to the last page of a PDF document.

You can enable/disable page navigation option in PDF Viewer using the following code snippet.,

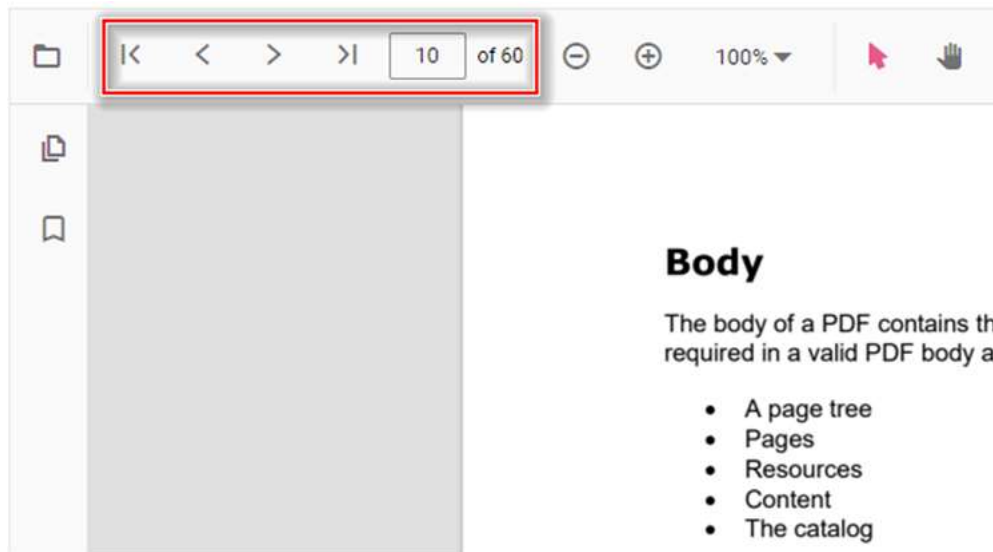
STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enableNavigation]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableNavigation]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
```

```
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```



Bookmark navigation

The Bookmarks saved in PDF files are loaded and made ready for easy navigation.

You can enable/disable bookmark navigation by using the following code snippet.,

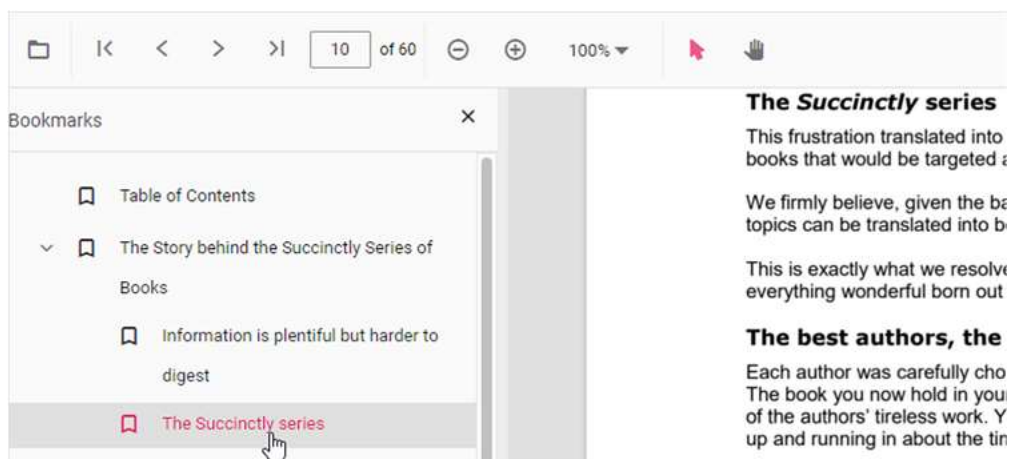
STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enableBookmark]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
```

```
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableBookmark]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
}
```



Thumbnail navigation

Thumbnails is the miniature representation of actual pages in PDF files. This feature displays thumbnails of the pages and allows navigation.

You can enable/disable thumbnail navigation by using the following code snippet.,

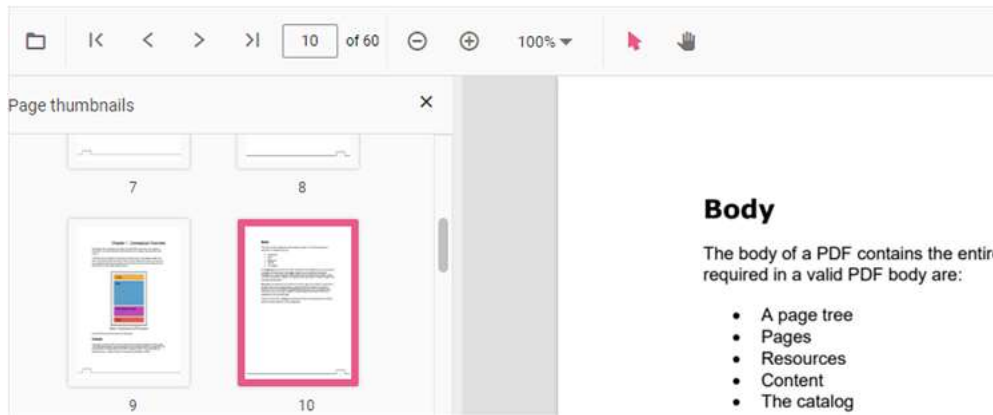
STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enableThumbnail]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService,TextSearchService,TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableThumbnail]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService,TextSearchService,TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
```

```
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```



Hyperlink navigation

Hyperlink navigation features enables navigation to the URLs (website links) in a PDF file.

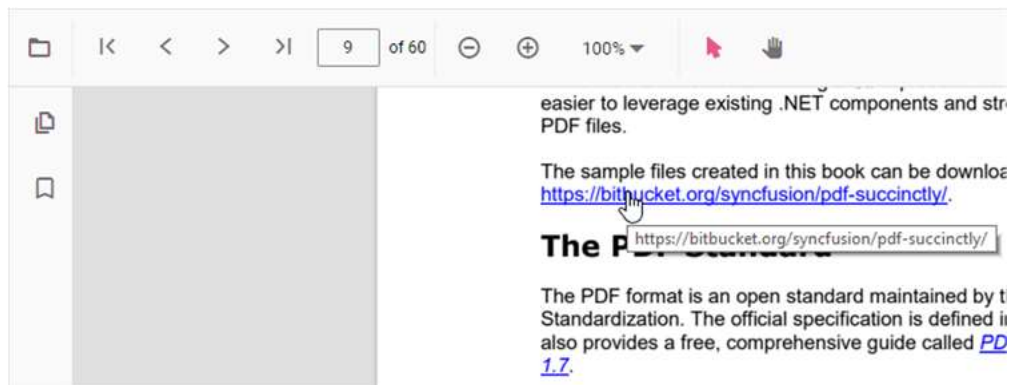


Table of content navigation

Table of contents navigation allows users to navigate to different parts of a PDF file that are listed in the table of contents section.

You can enable/disable link navigation by using the following code snippet.,

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'`
```

```
[enableHyperlink]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableHyperlink]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

You can change the open state of the hyperlink in the PDF Viewer by using the following code snippet,

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
```

```

} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer id="pdfViewer"
      [documentPath]='document'
      [hyperlinkOpenState]='linkOpenState'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    AnnotationService, TextSearchService, TextSelectionService,
    PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
  public linkOpenState = 'NewTab';
}

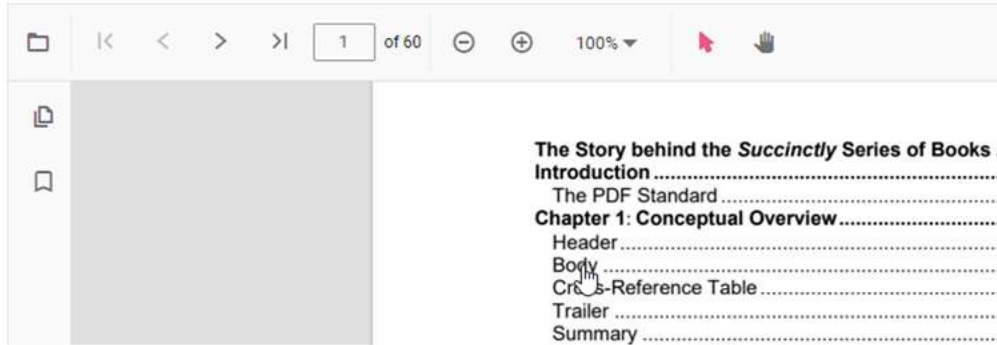
```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
  ThumbnailViewService, ToolbarService, NavigationService,
  TextSearchService, AnnotationService, TextSelectionService,
  PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer id="pdfViewer"
      [serviceUrl]='service'
      [documentPath]='document'
      [hyperlinkOpenState]='linkOpenState'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    AnnotationService, TextSearchService, TextSelectionService,
    PrintService]
})
export class AppComponent implements OnInit {
  public service =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
  public linkOpenState = 'NewTab';
}

```

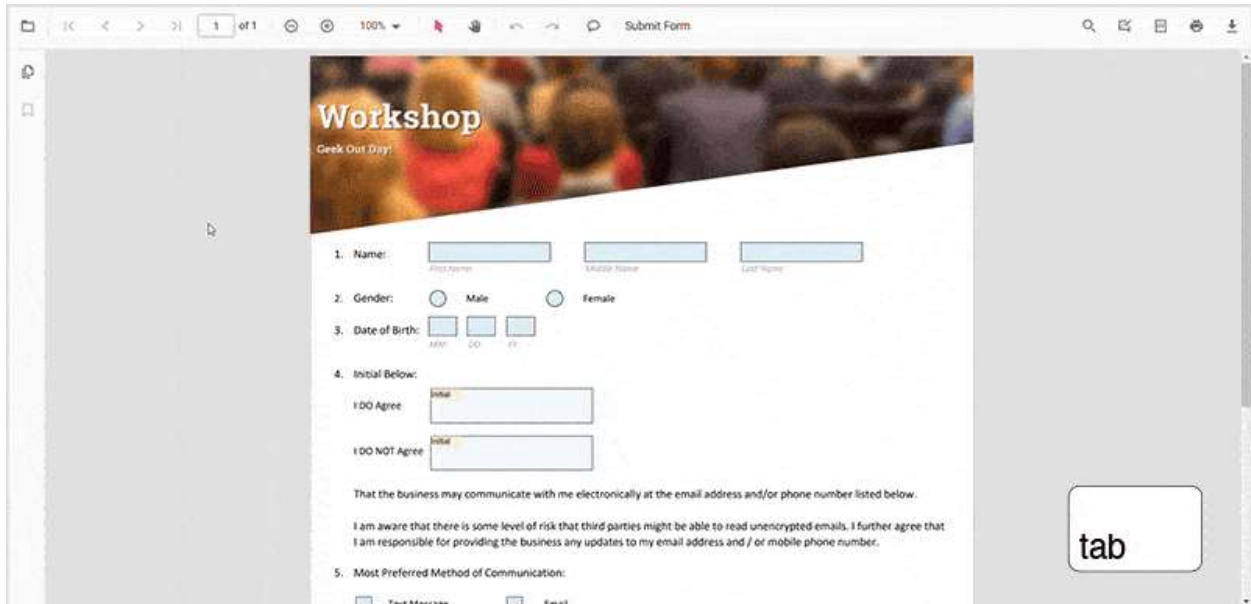


Keyboard navigation with Tab and Shift+Tab keys

The Syncfusion PDF Viewer component supports keyboard navigation for the form fields in a PDF document using the **Tab** and **Shift+Tab** keys. This allows users to easily navigate through the form fields in a PDF document, making it easier for them to fill out forms.

Here's how it works:

Pressing the **Tab** key will move the focus to the **next formfield** in the document.



Pressing **Shift+Tab** will move the focus to the **previous formfield** in the document.

1 of 1 100% Submit Form

4. Initial Below:

I DO Agree

I DO NOT Agree

That the business may communicate with me electronically at the email address and/or phone number listed below.

I am aware that there is some level of risk that third parties might be able to read unencrypted emails. I further agree that I am responsible for providing the business any updates to my email address and / or mobile phone number.

5. Most Preferred Method of Communication:

☐ Text Message ☐ Email

6. I would Like to Receive:

☐ Appointment Reminders ☐ Information Regarding Billing

☐ Requests for Customer Satisfaction reviews

7. Contact Information:

My Email My Phone

8. Signature:

Sign Date of Signature

shift + tab

Note: The order of the form fields is determined by the order in which they appear in a PDF document.

See also

- [Toolbar items](#)
- [Feature Modules](#)

Magnification in Angular PDF Viewer component

The magnification tools of the PDF Viewer contains ZoomIn, ZoomOut, Zoom, FitPage, and FitWidth tools in the

default toolbar. The PDF Viewer also has an option to show or hide the magnification tools in the default toolbar.

The following code snippet describes how to enable the magnification in PDF Viewer.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
enableMagnification="true"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
```

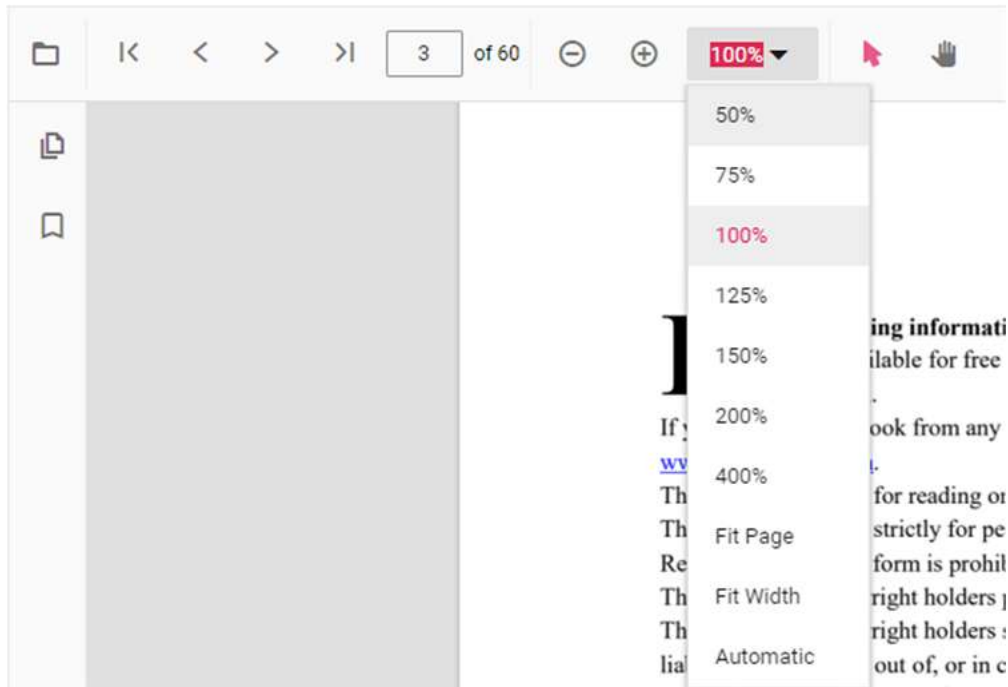
```
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
enableMagnification="true"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

The following magnification options are available in the default toolbar of PDF Viewer,

- **ZoomIn**:- Zoom in from the current zoom value of PDF pages.
- **ZoomOut**:- Zoom out from the current zoom value of PDF pages.
- **Zoom**:- Zoom to specific zoom value of PDF pages.
- **FitPage**:- Fits the page width with in the available view port size.
- **FitWidth**:- Fits the view port width based on the page content size.
- **Auto**:- Fits the page content with-in the viewport on resizing action.



PDF Viewer can support the zoom value ranges from 50 to 400.

See also

- [Toolbar items](#)
- [Feature Modules](#)

Text Search in Angular PDF Viewer component

The Text Search option in PDF Viewer is used to find and highlight the text content from the document. You can enable/disable the text search using the following code snippet.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enableTextSearch]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
```



```

AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

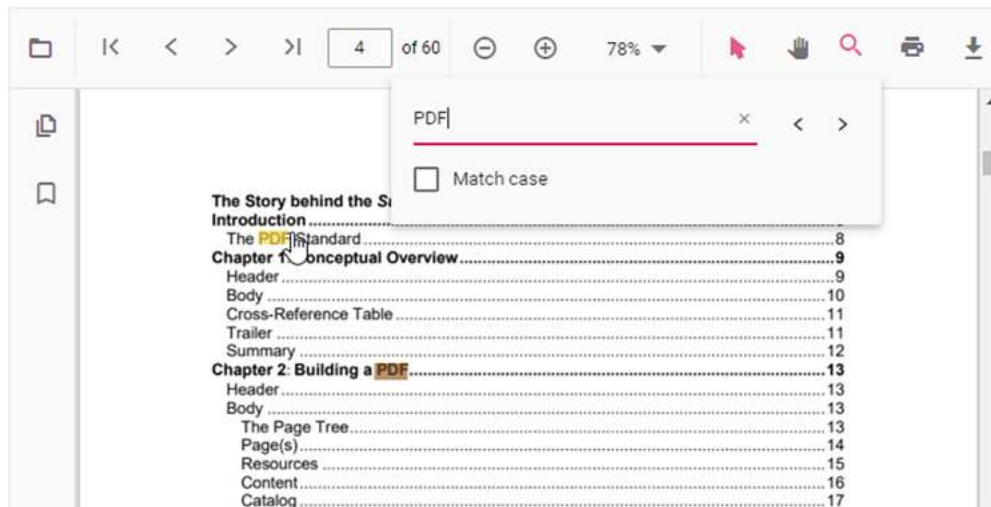
```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableTextSearch]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

```



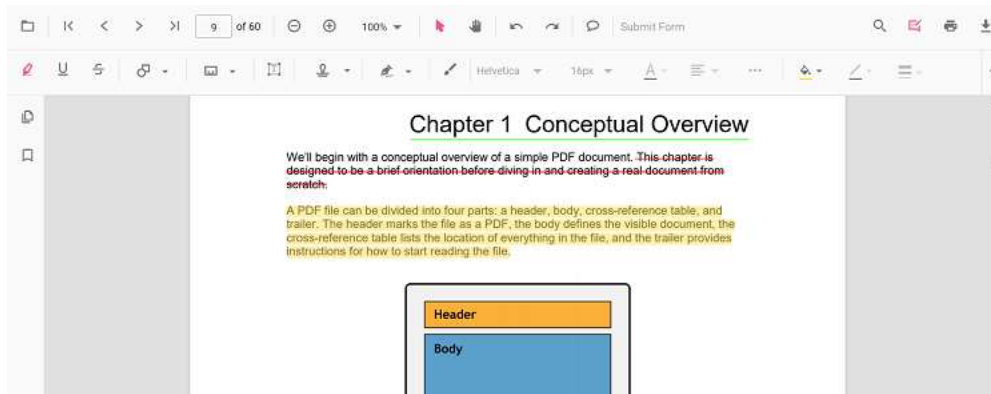
See also

- [Toolbar items](#)
- [Feature Modules](#)

Annotation

Text Markup Annotation in the Angular PDF Viewer component

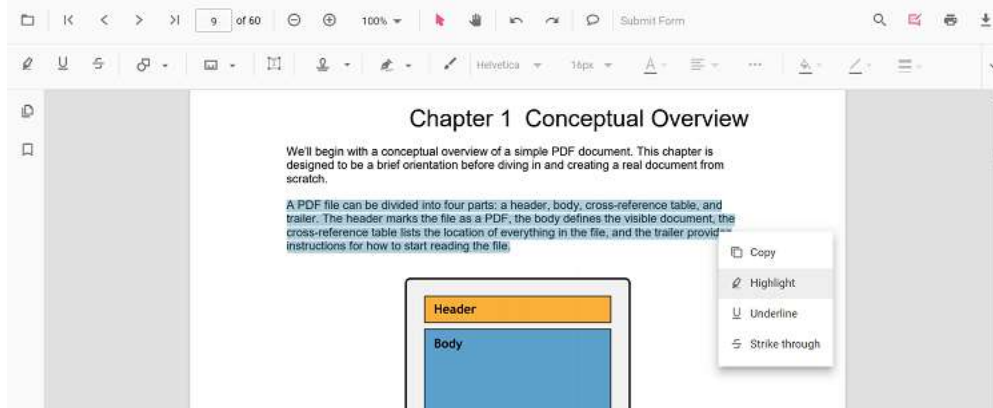
The PDF Viewer control provides the options to add, edit, and delete text markup annotations such as highlight, underline, and strikethrough annotations in the PDF document.



Highlight a text

There are two ways to highlight a text in the PDF document:

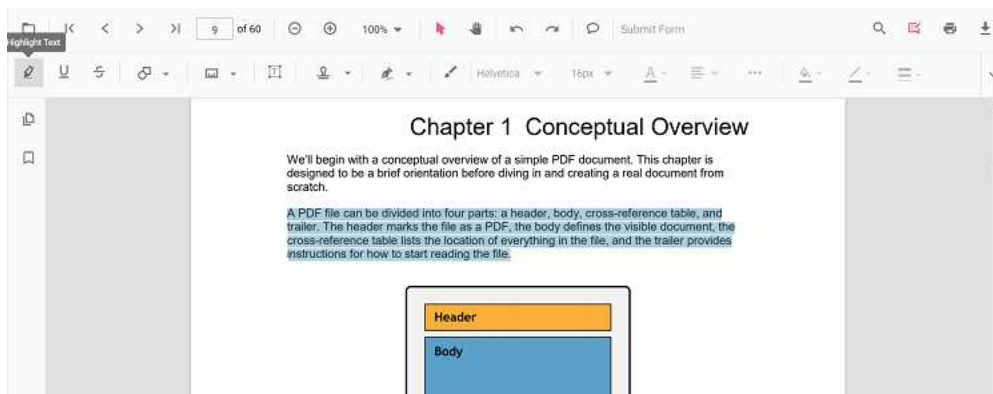
1. Using the context menu
 - Select a text in the PDF document and right-click it.
 - Select **Highlight** option in the context menu that appears.



<!-- markdownlint-disable MD029 -->

2. Using the annotation toolbar

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Highlight** button in the annotation toolbar. It enables the highlight mode.
- Select the text and the highlight annotation will be added.
- You can also select the text and apply the highlight annotation using the **Highlight** button.



In the pan mode, if the highlight mode is entered, the PDF Viewer control will switch to text select mode to enable the text selection for highlighting the text.

Refer to the following code sample to switch to the highlight mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Highlight</button>
```

```

<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Highlight');
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Highlight</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {

```

```

public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Highlight');
}
}

```

Refer to the following code sample to switch back to normal mode from the highlight mode.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Highlight Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Highlight');
}
setNone() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];

```

```
pdfviewer.annotationModule.setAnnotationMode("None");
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Highlight Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode('Highlight');
  }
  setNone() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode("None");
  }
}
```

Highlight a text programmatically

The PDF Viewer library enables you to programmatically highlight text within the PDF Viewer control using the [addAnnotation\(\)](#) method.

Here's an example of how you can use the **addAnnotation()** method to apply highlighting programmatically:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, HighlightSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Highlight</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Highlight", {
      bounds: [{ x: 97, y: 110, width: 350, height: 14 }],
      pageNumber: 1
    } as HighlightSettings);
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, HighlightSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
```

```

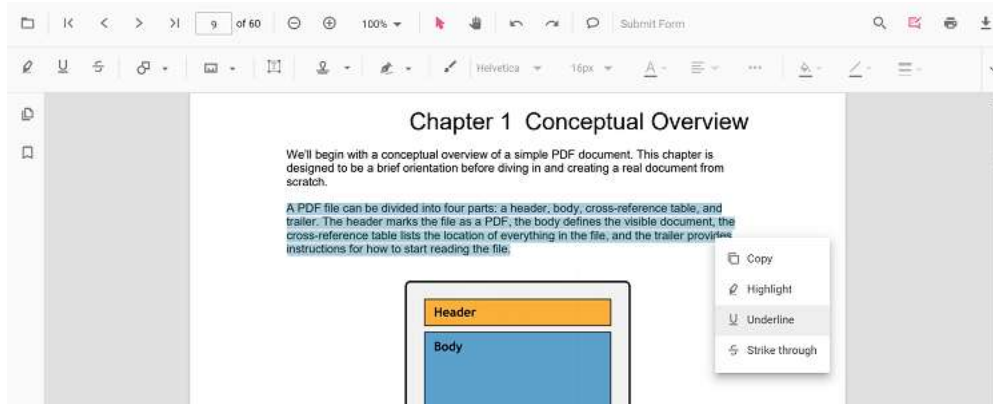
@Component ({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <button (click)="addAnnotation()">Highlight</button>
    <ejs-pdfviewer
      id="pdfViewer"
      [documentPath]='document'
      [serviceUrl]='service'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService,
    PageOrganizerService]
  })
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
  public service: string =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Highlight", {
      bounds: [{ x: 97, y: 110, width: 350, height: 14 }],
      pageNumber: 1
    } as HighlightSettings);
  }
}

```

Underline a text

There are two ways to underline a text in the PDF document:

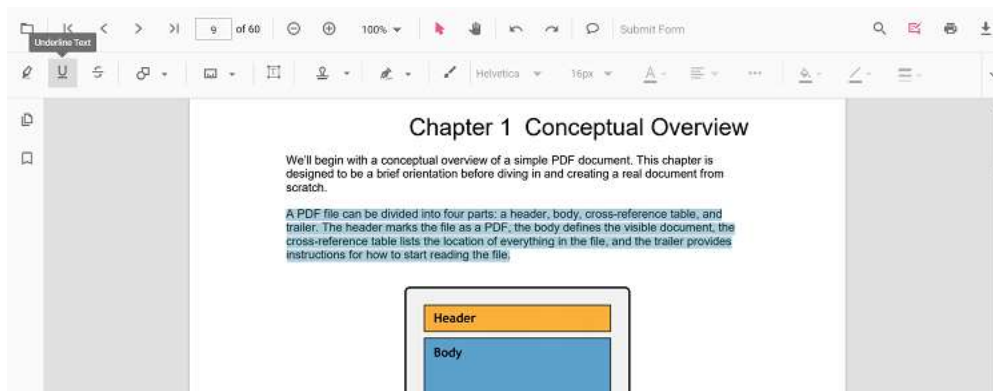
1. Using the context menu
 - Select a text in the PDF document and right-click it.
 - Select the **Underline** option in the context menu that appears.



<!-- markdownlint-disable MD029 -->

2. Using the annotation toolbar

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Underline** button in the annotation toolbar. It enables the underline mode.
- Select the text and the underline annotation will be added.
- You can also select the text and apply the underline annotation using the **Underline** button.



In the pan mode, if the underline mode is entered, the PDF Viewer control will switch to text select mode to enable the text selection for underlining the text.

Refer to the following code sample to switch to the underline mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Underline</button>
```

```

<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Underline');
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Underline</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {

```

```

public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Underline');
}
}

```

Refer to the following code sample to switch back to normal mode from the underline mode.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Underline Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Underline');
}
setNone() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];

```

```
pdfviewer.annotationModule.setAnnotationMode("None");
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Underline Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode('Underline');
  }
  setNone() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode("None");
  }
}
```

Underline a text programmatically

The PDF Viewer library enables you to programmatically Underline text within the PDF Viewer control using the [addAnnotation\(\)](#) method.

Here's an example of how you can use the **addAnnotation()** method to apply Underline programmatically:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, UnderlineSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Underline text</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Underline", {
      bounds: [{ x: 250, y: 148, width: 345, height: 14 }],
      pageNumber: 2
    } as HighlightSettings);
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, UnderlineSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
```

```

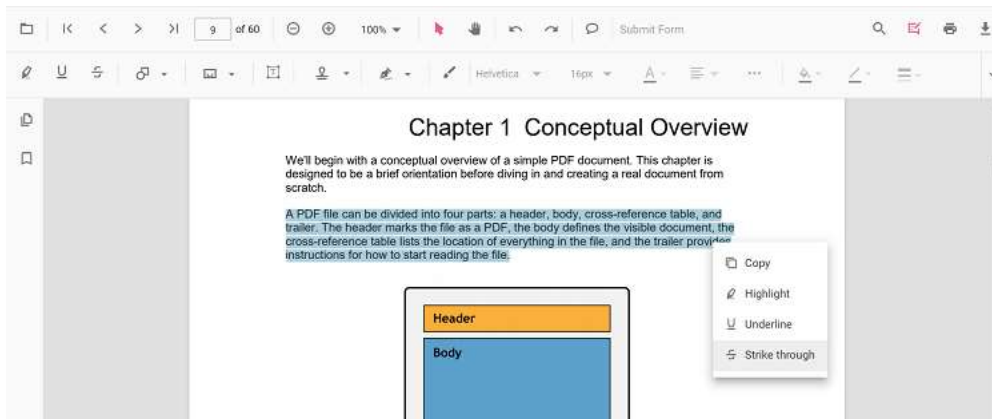
@Component ({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <button (click)="addAnnotation()">Underline text</button>
    <ejs-pdfviewer
      id="pdfViewer"
      [documentPath]='document'
      [serviceUrl]='service'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService,
    PageOrganizerService]
  })
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
  succinctly.pdf';
  public service: string =
  'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Underline", {
      bounds: [{ x: 250, y: 148, width: 345, height: 14 }],
      pageNumber: 2
    } as HighlightSettings);
  }
}

```

Strikethrough a text

There are two ways to strikethrough a text in the PDF document:

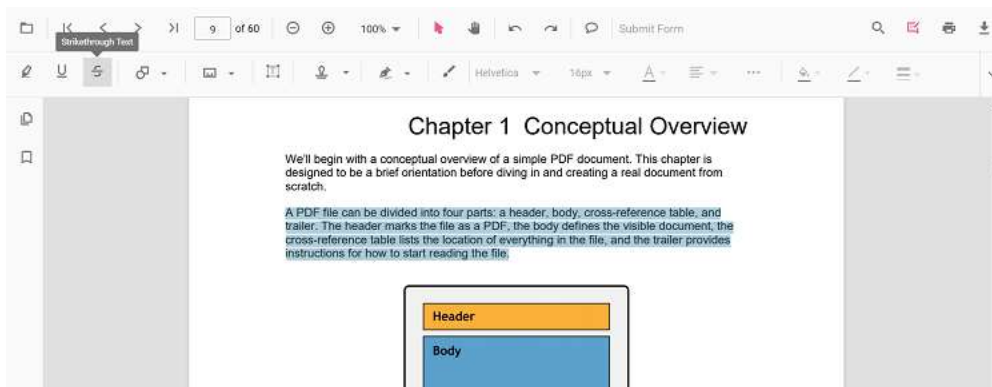
1. Using the context menu
 - Select a text in the PDF document and right-click it.
 - Select the **Strikethrough** option in the context menu that appears.



<!-- markdownlint-disable MD029 -->

2. Using the annotation toolbar

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Strikethrough** button in the annotation toolbar. It enables the strikethrough mode.
- Select the text and the strikethrough annotation will be added.
- You can also select the text and apply the strikethrough annotation using the **Strikethrough** button.



Note: While you're in the pan mode, for navigating through the document, and you click on the strikethrough button, the PDF Viewer control will smoothly transition to text select mode. This seamless transition ensures a fluid experience when switching between different interaction modes within the PDF Viewer interface.

Refer to the following code sample to switch to strikethrough mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
```

```

selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough text</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Strikethrough');
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough text</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,

```



```

AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode('Strikethrough');
  }
}

```

Refer to the following code sample to switch back to normal mode from the strikethrough mode.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];

```

```
pdfviewer.annotationModule.setAnnotationMode('Strikethrough');
}
setNone() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("None");
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough Text</button>
<button (click)="setNone()">None</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Strikethrough');
}
setNone() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("None");
}
}
```

Strikethrough a text programmatically

The PDF Viewer library enables you to programmatically Strikethrough text within the PDF Viewer control using the [addAnnotation\(\)](#) method.

Here's an example of how you can use the **addAnnotation()** method to apply Strikethrough programmatically:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StrikethroughSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough text</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Strikethrough", {
      bounds: [{ x: 250, y: 144, width: 345, height: 14 }],
      pageNumber: 2
    } as StrikethroughSettings);
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
```

```

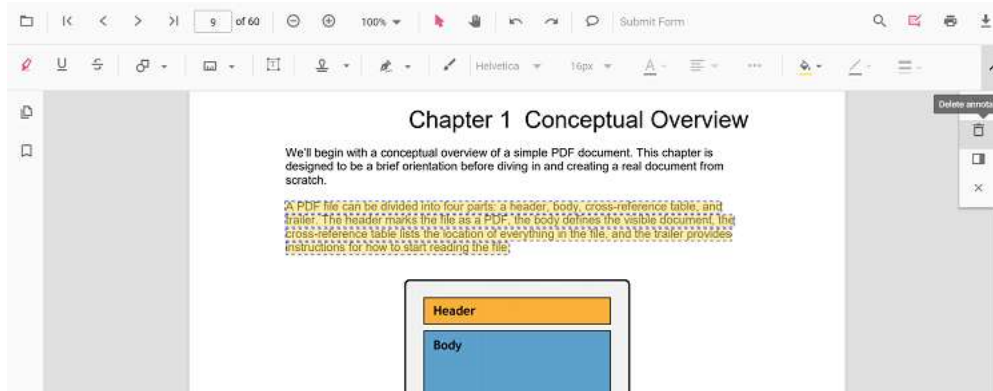
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StrikethroughSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Strikethrough text</button>
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]='document'
  [serviceUrl]='service'
  style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Strikethrough", {
      bounds: [{ x: 250, y: 144, width: 345, height: 14 }],
      pageNumber: 2
    } as StrikethroughSettings);
  }
}

```

Deleting a text markup annotation

The selected annotation can be deleted in the following ways:

1. Using the Delete key
 - Select the annotation to be deleted.
 - Click the Delete key in the keyboard. The selected annotation will be deleted.
2. Using the annotation toolbar
 - Select the annotation to be deleted.
 - Click the **Delete Annotation** button in the annotation toolbar. The selected annotation will be deleted.

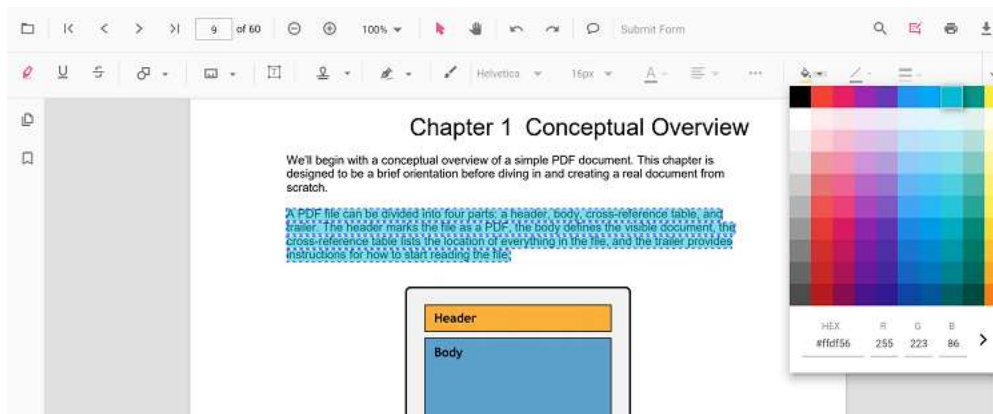


Editing the properties of the text markup annotation

The color and the opacity of the text markup annotation can be edited using the Edit Color tool and the Edit Opacity tool in the annotation toolbar.

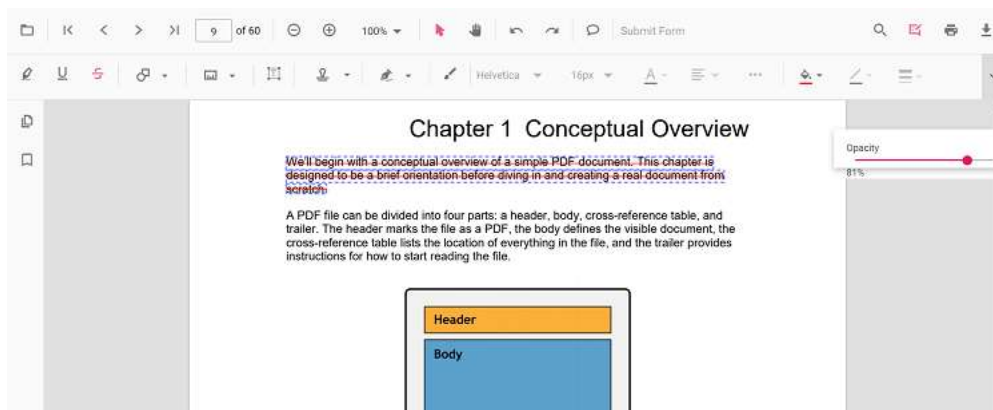
Editing color

The color of the annotation can be edited using the color palette provided in the Edit Color tool.



Editing opacity

The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



Setting default properties during the control initialization

The properties of the text markup annotation can be set before creating the control using the highlightSettings, underlineSettings, and strikethroughSettings.

After editing the default color and opacity using the Edit Color tool and Edit Opacity tool, they will be changed to the selected values.

Refer to the following code sample to set the default annotation settings.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
[highlightSettings]='highlightSettings'
[underlineSettings]='underlineSettings'
[strikethroughSettings]='strikethroughSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
public highlightSettings = { author: 'Guest User', subject: 'Important',
color: '#ffff00', opacity: 0.9, modifiedDate: '' };
public underlineSettings = { author: 'Guest User', subject: 'Points to be
remembered', color: '#00ffff', opacity: 0.9, modifiedDate: '' };
public strikethroughSettings = { author: 'Guest User', subject: 'Not
Important', color: '#ff00ff', opacity: 0.9, modifiedDate: '' };
ngOnInit(): void {
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
```

```

AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
[highlightSettings]='highlightSettings'
[underlineSettings]='underlineSettings'
[strikethroughSettings]='strikethroughSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public highlightSettings = { author: 'Guest User', subject: 'Important',
color: '#ffff00', opacity: 0.9, modifiedDate: '' };
  public underlineSettings = { author: 'Guest User', subject: 'Points to be
remembered', color: '#00ffff', opacity: 0.9, modifiedDate: '' };
  public strikethroughSettings = { author: 'Guest User', subject: 'Not
Important', color: '#ff00ff', opacity: 0.9, modifiedDate: '' };
  ngOnInit(): void {
  }
}

```

Performing undo and redo

The PDF Viewer performs undo and redo for the changes made in the PDF document. In the text markup annotation, undo and redo actions are provided for:

- Inclusion of the text markup annotations.
- Deletion of the text markup annotations.
- Change of either color or opacity of the text markup annotations.

The undo and redo actions can be done in the following ways:

1. Using the keyboard shortcuts:

After performing a text markup annotation action, you can undo it by using the Ctrl + Z shortcut and redo by using the Ctrl + Y shortcut.

2. Using toolbar:

The Undo and redo can be done using the **Undo** tool and **Redo** tool provided in the toolbar.

Refer to the following code sample for calling undo and redo actions from the client-side.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="Undo()">Undo</button>
<button (click)="Redo()">Redo</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
Undo() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.undo();
}
Redo() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.redo();
}
}
```

SERVER-BACKED

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



```

MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="Undo()">Undo</button>
<button (click)="Redo()">Redo</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  Undo() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.undo();
  }
  Redo() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.redo();
  }
}

```

Saving the text markup annotation

When you click the download tool in the toolbar, the text markup annotations will be saved in the PDF document. This action will not affect the original document.

Printing the text markup annotation

When the print tool is selected in the toolbar, the PDF document will be printed along with the text markup annotations added to the pages. This action will not affect the original document.

Disabling text markup annotation

The PDF Viewer control provides an option to disable the text markup annotation feature. The code sample for disabling the feature is as follows.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]='document'
  [resourceUrl]='resource'
  [enableTextMarkupAnnotation]='false'
  style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]='document'
  [serviceUrl]='service'
  [enableTextMarkupAnnotation]='false'
  style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,

```

```

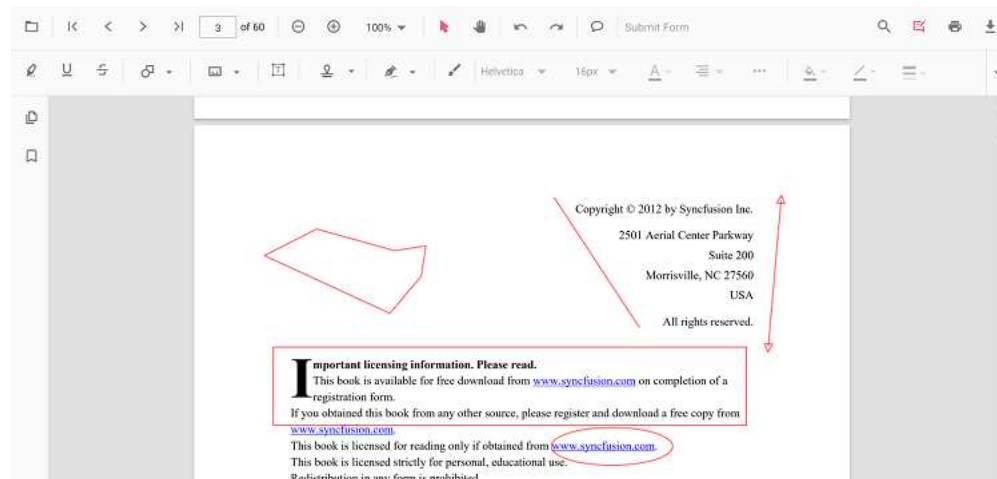
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
}

```

Shape Annotation in Angular PDF Viewer component

The PDF Viewer control provides the options to add, edit, and delete the shape annotations. The shape annotation types supported in the PDF Viewer control are:

- Line
- Arrow
- Rectangle
- Circle
- Polygon

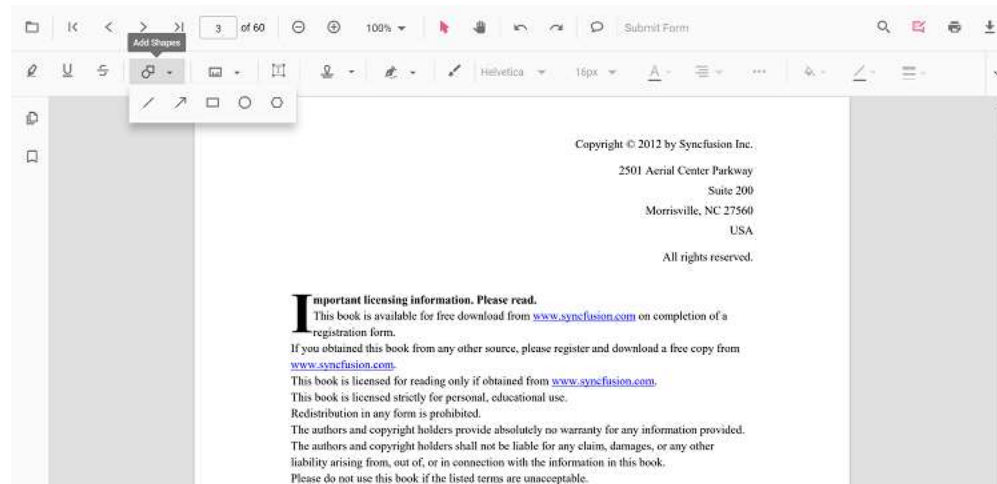


Adding a shape annotation to the PDF document

Shape annotations can be added to the PDF document using the annotation toolbar.

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Click the **Shape Annotation** drop-down button. A drop-down pop-up will appear and shows the shape annotations to be added.
- Select the shape types to be added to the page in the drop-down pop-up. It enables the selected shape annotation mode.
- You can add the shapes over the pages of the PDF document.

In the pan mode, if the shape annotation mode is entered, the PDF Viewer control will switch to text select mode.



Refer to the following code sample to switch to the circle annotation mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// Specifies the template string for the PDF Viewer component.
template: `<button (click)="addAnnotation()">Circle</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl] = 'resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("Circle");
}
}
```

```
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<button (click)="addAnnotation()">Circle</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode("Circle");
  }
}
```

Adding a shape annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a shape annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's a example of how you can utilize the **addAnnotation()** method to include a shape annotation programmatically:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, LineSettings, ArrowSettings,
```

```

RectangleSettings, CircleSettings, PolygonSettings } from '@syncfusion/ej2-
angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <button (click)="addLineAnnotation()">Add Line annotation
    Programmatically</button>
    <button (click)="addArrowAnnotation()">Add Arrow annotation
    Programmatically</button>
    <button (click)="addRectangleAnnotation()">Add Rectangle annotation
    Programmatically</button>
    <button (click)="addCircleAnnotation()">Add Circle annotation
    Programmatically</button>
    <button (click)="addPolygonAnnotation()">Add Polygon annotation
    Programmatically</button>
    <ejs-pdfviewer
    id="pdfViewer"
    [documentPath]='document'
    [resourceUrl]='resource'
    style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
  MagnificationService,
  ThumbnailViewService, ToolbarService, NavigationService,
  TextSearchService, TextSelectionService, PrintService,
  AnnotationService, FormDesignerService, FormFieldsService,
  PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
  succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
  pdfviewer-lib";
  ngOnInit(): void {
  }
  addLineAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Line", {
      offset: { x: 200, y: 230 },
      pageNumber: 1,
      vertexPoints: [{ x: 200, y: 230 }, { x: 350, y: 230 }]
    } as LineSettings);
  }
  addArrowAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Arrow", {
      offset: { x: 200, y: 370 },
      pageNumber: 1,
      vertexPoints: [{ x: 200, y: 370 }, { x: 350, y: 370 }]
    } as ArrowSettings);
  }
  addRectangleAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Rectangle", {
      offset: { x: 200, y: 480 },

```

```

pageNumber: 1,
width: 150,
height: 75
} as RectangleSettings);
}
addCircleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Circle", {
offset: { x: 200, y: 620 },
pageNumber: 1,
width: 90,
height: 90
} as CircleSettings);
}
addPolygonAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Polygon", {
offset: { x: 200, y: 800 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 800 }, { x: 242, y: 771 }, { x: 289, y: 799 }, {
x: 278, y: 842 }, { x: 211, y: 842 }, { x: 200, y: 800 }]
} as PolygonSettings);
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, LineSettings, ArrowSettings,
RectangleSettings, CircleSettings, PolygonSettings } from '@syncfusion/ej2-
angular-pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addLineAnnotation()">Add Line annotation
Programmatically</button>
<button (click)="addArrowAnnotation()">Add Arrow annotation
Programmatically</button>
<button (click)="addRectangleAnnotation()">Add Rectangle annotation
Programmatically</button>
<button (click)="addCircleAnnotation()">Add Circle annotation
Programmatically</button>
<button (click)="addPolygonAnnotation()">Add Polygon annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,

```

```

providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addLineAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Line", {
offset: { x: 200, y: 230 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 230 }, { x: 350, y: 230 }]
} as LineSettings);
}
addArrowAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Arrow", {
offset: { x: 200, y: 370 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 370 }, { x: 350, y: 370 }]
} as ArrowSettings);
}
addRectangleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Rectangle", {
offset: { x: 200, y: 480 },
pageNumber: 1,
width: 150,
height: 75
} as RectangleSettings);
}
addCircleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Circle", {
offset: { x: 200, y: 620 },
pageNumber: 1,
width: 90,
height: 90
} as CircleSettings);
}
addPolygonAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Polygon", {
offset: { x: 200, y: 800 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 800 }, { x: 242, y: 771 }, { x: 289, y: 799 }, {
x: 278, y: 842 }, { x: 211, y: 842 }, { x: 200, y: 800 }]
} as PolygonSettings);
}
}

```



```
}

```

Edit the existing shape annotation programmatically

To modify existing shape annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="editLineAnnotation()">Edit Line annotation
Programmatically</button>
<button (click)="editArrowAnnotation()">Edit Arrow annotation
Programmatically</button>
<button (click)="editRectangleAnnotation()">Edit Rectangle annotation
Programmatically</button>
<button (click)="editCircleAnnotation()">Edit Circle annotation
Programmatically</button>
<button (click)="editPolygonAnnotation()">Edit Polygon annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  editLineAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
      if (pdfviewer.annotationCollection[i].subject === "Line") {
```

```

pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editArrowAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Arrow") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editRectangleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Rectangle") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editCircleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Circle") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editPolygonAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Polygon") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}

```

```

}
}
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editLineAnnotation()">Edit Line annotation
Programmatically</button>
<button (click)="editArrowAnnotation()">Edit Arrow annotation
Programmatically</button>
<button (click)="editRectangleAnnotation()">Edit Rectangle annotation
Programmatically</button>
<button (click)="editCircleAnnotation()">Edit Circle annotation
Programmatically</button>
<button (click)="editPolygonAnnotation()">Edit Polygon annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
editLineAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Line") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;

```

```

pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editArrowAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Arrow") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editRectangleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Rectangle") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editCircleAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Circle") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editPolygonAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Polygon") {
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}

```

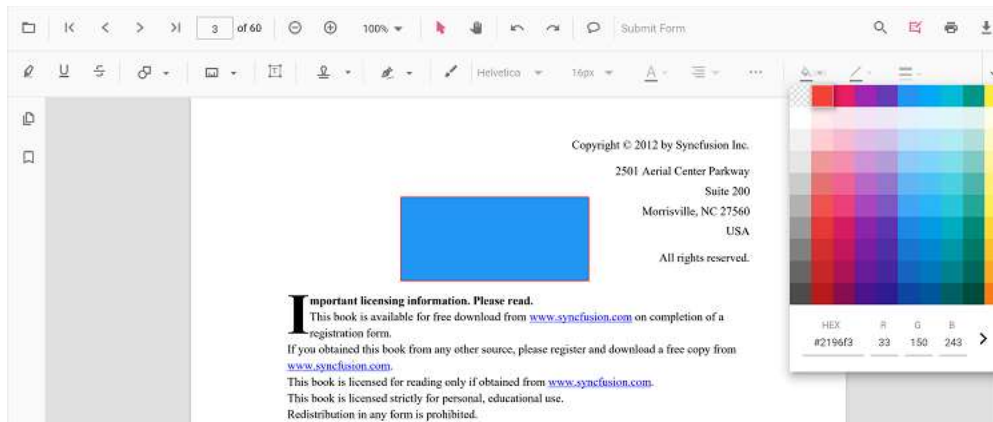
```
}  
}
```

Editing the properties of the shape annotation

The fill color, stroke color, thickness, and opacity of the shape annotation can be edited using the Edit color tool, Edit stroke color tool, Edit thickness tool, and Edit opacity tool in the annotation toolbar.

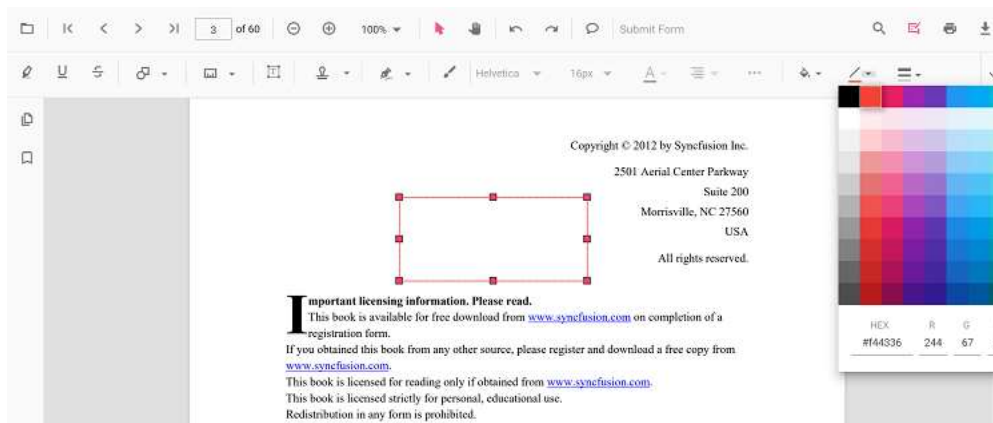
Editing fill color

The fill color of the annotation can be edited using the color palette provided in the Edit Color tool.



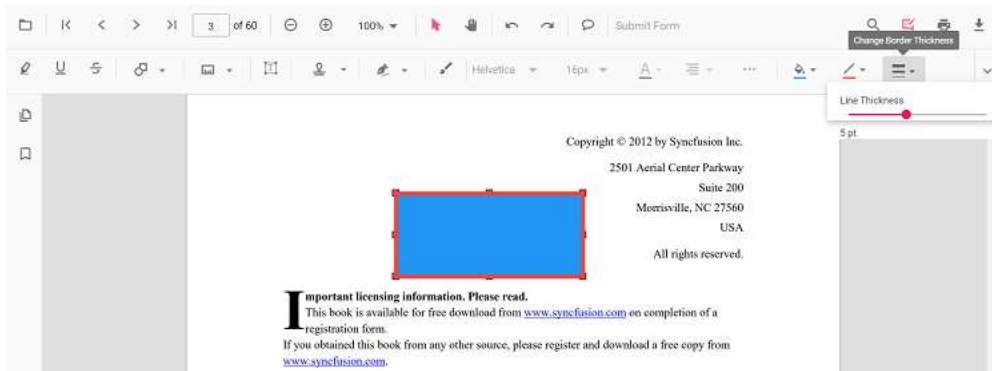
Editing stroke color

The stroke color of the annotation can be edited using the color palette provided in the Edit Stroke Color tool.



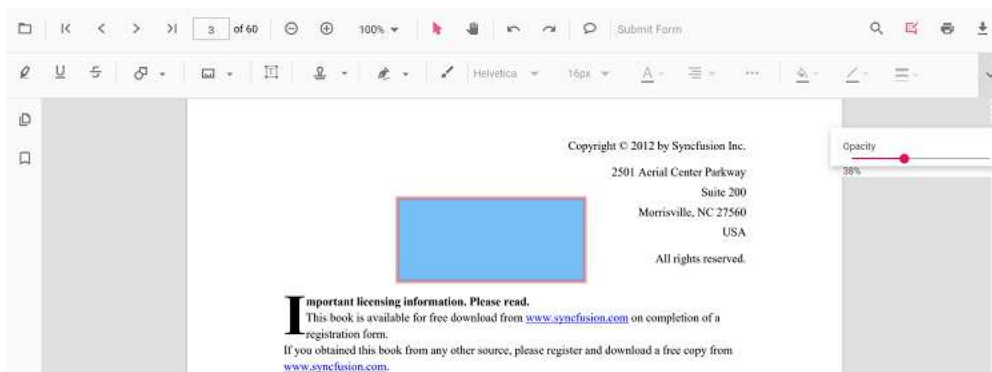
Editing thickness

The thickness of the border of the annotation can be edited using the range slider provided in the Edit Thickness tool.



Editing opacity

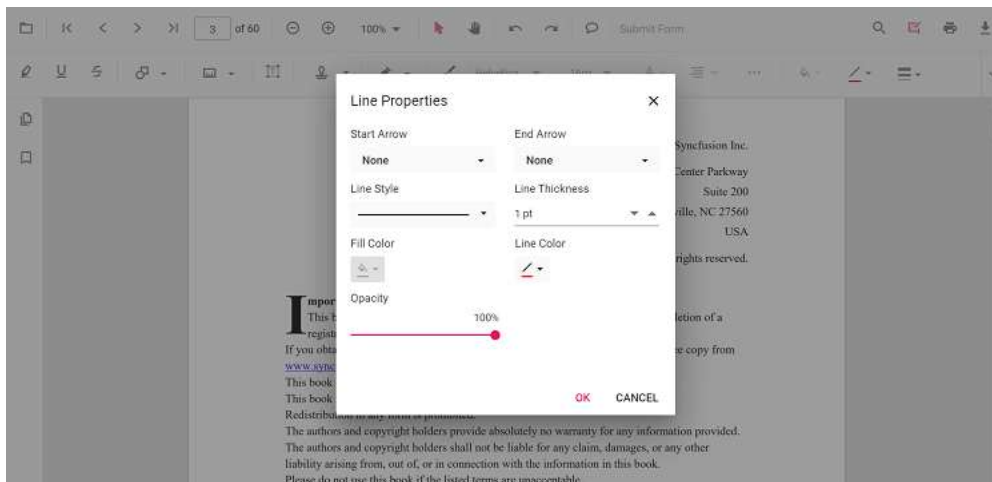
The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



Editing the line properties

The properties of the line shapes such as line and arrow annotations can be edited using the Line Properties window. It can be opened by selecting the Properties option in the context menu that appears on right-clicking the line and arrow annotations.

Refer to the following code sample to set the default annotation settings.



Setting default properties during the control initialization

The properties of the shape annotations can be set before creating the control using LineSettings, ArrowSettings, RectangleSettings, CircleSettings, and PolygonSettings.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[lineSettings]='lineSettings'
[arrowSettings]='arrowSettings'
[rectangleSettings]='rectangleSettings'
[circleSettings]='circleSettings'
[polygonSettings]='polygonSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public lineSettings = { fillColor: 'blue', opacity: 0.6, strokeColor: 'green'
};
  public arrowSettings = { fillColor: 'green', opacity: 0.6, strokeColor:
'blue' };
  public rectangleSettings = { fillColor: 'yellow', opacity: 0.6, strokeColor:
'orange' };
  public circleSettings = { fillColor: 'orange', opacity: 0.6, strokeColor:
'pink' };
  public polygonSettings = { fillColor: 'pink', opacity: 0.6, strokeColor:
'yellow' };
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'

```

```

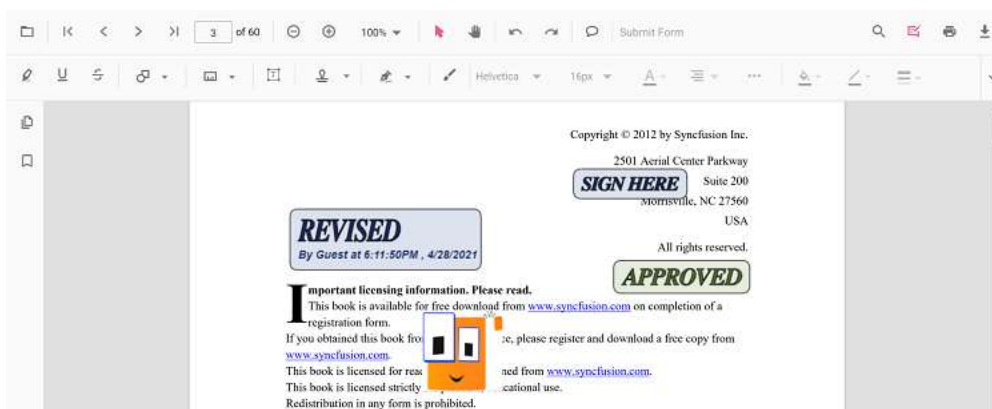
[lineSettings]='lineSettings'
[arrowSettings]='arrowSettings'
[rectangleSettings]='rectangleSettings'
[circleSettings]='circleSettings'
[polygonSettings]='polygonSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public lineSettings = { fillColor: 'blue', opacity: 0.6, strokeColor: 'green'
};
public arrowSettings = { fillColor: 'green', opacity: 0.6, strokeColor:
'blue' };
public rectangleSettings = { fillColor: 'yellow', opacity: 0.6, strokeColor:
'orange' };
public circleSettings = { fillColor: 'orange', opacity: 0.6, strokeColor:
'pink' };
public polygonSettings = { fillColor: 'pink', opacity: 0.6, strokeColor:
'yellow' };
}

```

Stamp Annotation in Angular PDF Viewer component

The PDF Viewer control provides options to add, edit, delete, and rotate the following stamp annotation in the PDF documents:

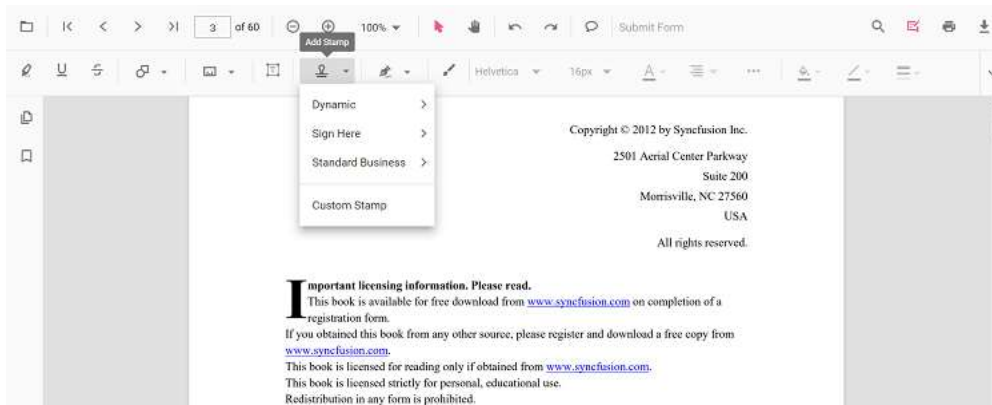
- Dynamic
- Sign Here
- Standard Business
- Custom Stamp



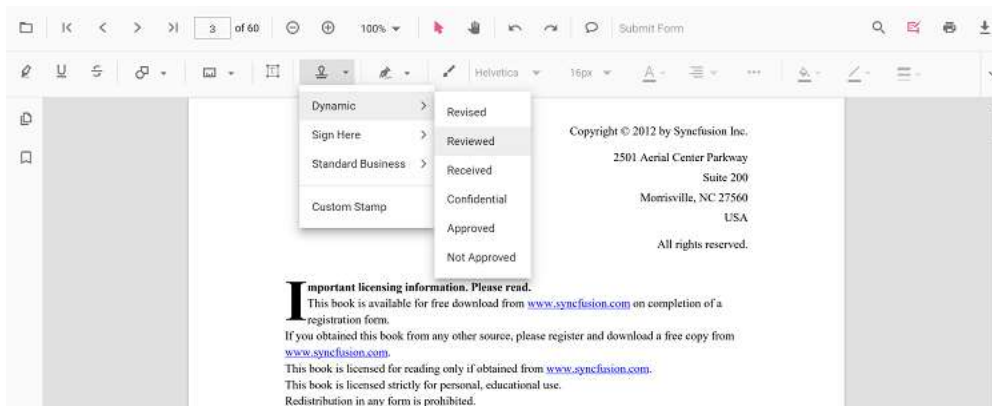
Adding stamp annotations to the PDF document

The stamp annotations can be added to the PDF document using the annotation toolbar.

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Click the **Stamp Annotation** drop-down button. A drop-down pop-up will appear and shows the stamp annotations to be added.



- Select the annotation type to be added to the page in the pop-up.



- You can add the annotation over the pages of the PDF document.

In the pan mode, if the stamp annotation mode is entered, the PDF Viewer control will switch to text select mode.

Refer to the following code sample to switch to the stamp annotation mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, SignStampItem
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
```

```
// Specifies the template string for the PDF Viewer component.
template: `<button (click)="addAnnotation()">Stamp</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("Stamp", null,
SignStampItem.Witness);
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, SignStampItem
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// Specifies the template string for the PDF Viewer component.
template: `<button (click)="addAnnotation()">Stamp</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
```

```

public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("Stamp", null,
SignStampItem.Witness);
}
}

```

Adding a Stamp annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a Stamp annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's an example of how you can utilize the **addAnnotation()** method to include a Stamp annotation programmatically:

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StampSettings, DynamicStampItem ,
SignStampItem, StandardBusinessStampItem, CustomStampSettings} from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addDynamicStamp()">Add Dynamic Stamp
Programmatically</button>
<button (click)="addSignStamp()">Add Sign Stamp Programmatically</button>
<button (click)="addStandardBusinessStamp()">Add StandardBusiness Stamp
Programmatically</button>
<button (click)="addCustomStamp()">Add Custom Stamp Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {

```

```
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-  
succinctly.pdf';  
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-  
pdfviewer-lib";  
ngOnInit(): void {  
}  
addDynamicStamp() {  
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];  
pdfviewer.annotation.addAnnotation("Stamp", {  
offset: { x: 200, y: 140 },  
pageNumber: 1  
} as StampSettings, DynamicStampItem.Approved);  
}  
addSignStamp() {  
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];  
pdfviewer.annotation.addAnnotation("Stamp", {  
offset: { x: 200, y: 240 },  
pageNumber: 1  
} as StampSettings, undefined, SignStampItem.Witness);  
}  
addStandardBusinessStamp() {  
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];  
pdfviewer.annotation.addAnnotation("Stamp", {  
offset: { x: 200, y: 340 },  
pageNumber: 1  
} as StampSettings, undefined, undefined,  
StandardBusinessStampItem.Approved);  
}  
addCustomStamp() {  
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];  
viewer.annotation.addAnnotation('Stamp',  
{  
offset: { x: 100, y: 440 },  
width: 46,  
author: 'Guest',  
height: 100,  
isLock: true,  
pageNumber: 1,  
customStamps: [  
{  
customStampName: "Image",  
customStampImageSource:  
"data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAkGBwgHBGkIBWgKCgkLD  
RYPDQwMDRsUFRAWIB0iIiAdHx8kKDQsJCYxJx8fLT0tMTU3Ojo6Iys/RD84QzQ5OjcBCGoKDWNGg  
8PGjclHyU3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3N//  
AABEIAIIAQwMBIgACEQEDEQH/xAAbAAEAAGMBAQAAAAAAAAAAABQYBAwQHAv/EAEEQAAEDAwIE  
AwYDBAYLAAAAAAECAwQABREGIRIxQVETyXEHFCIygZEVQmIjUnKCJCuzU6HRFhc1c5KisbKzwvD/x  
AAVAQEBAAAAAAAAAAAAAAAAAAAf/EABQRAQAAAAAAAAAAAAAAAAAAD/2gAMAwEAAhEDEQA/AP  
caUpQKUppQKUppQKUppQKVzXGdftsN2ZPfbYjNJ4nHHdgJfFEK5Q5ttbuUaQhcNxvxUPcklPfFlQ  
dlYJxURpe/salthuMNpxEYvuNtKc28VKVcPGB2JB577VyZ7pNuUxy26eWlCml8Mu4OI4kR/0oB2Wv  
p2T17EJK43qDbloakOqL7m6I7TanHVjOMhCQTjzxgVut89i4Mqdj8Y4VlC00IKFIUOYKTuOn0INRZ  
ZtWkrVLuDpIIHHJlPK4nXl0DJR5kk4A5DYDarVoWbHuVgTPjvF5Ul5xx5zhIBC4jkJyBlI+UHqeOF  
jpSlApSlApSlApSlApSlApSlArClAczgVmqr7QZLptkezXHilKvD4ihxKsFprBU6v6IB+pFBT  
dUKflUuFuFa0WpyUIVoYBx706chchXdKEhZSPLNXxVTsOW6NdjNxivJS0iLEidHnDhLaPME4z5ZzVH  
k6kThulvTpyElJf8L3OwwlZDaGc4XJXjklXDhP6ULWd63ybrLlrqlmNalhLcAKEeQgcTbbvyuSchN  
w5KGweZJPIVRyoDT6okfSLnfWhmCWlu43FGAUKxu2j9atyT+UHvirZBixLzBaiQ2kR4zCMIQNZKRw  
uz2vLZ7czBqc4GWhl3KidvpR6qJJJPevOvaFcCXqC4HSgmzxlxObmvJJAPXwwe2M8R9R3FOclxde
```

```

qQcEW+C44jTFuVxPvtnHvCvI+e4HYZPavV4sdmLGajxmktMtJCENpGAKdKBUpbixRNO2dm3Q0/Cj4
lrPNazzUf/uWKlkkEZByKDNKUoFKUoFKUoFKUoFKwahZ2p7dFfMZhTs+ZnHu0FHirB/VjZHQogUE3
WOIYzUApzUlwBKUxLowQCFL/bv467DCEN6qr5i6btK5ht+Z0lXlCxlLkiTtr8whGG8fy0HdKlFZo
rymHbjH8dPNlC+NY/1Tk1XNTE0ml2SCXBFnrkOpX7uh6ItkKUBzPGEhzhjcAlbokKLAZS1BjMx20j
AQy2EjHoK85i6PuOovaFNv+pWPDt8J/ggMKUCXktq+BX8HNXmT2G9HLFl1trSyW2GrUFgbluT3eCIR
IS26tS/isjwgCcDl35Z3qBlSb/edVcN58e4tojKafiW2MfDQpRBXF8X5UnZPGsq5ZAr0TV2j52oL9
Anx7wqCxHYWypLbeXAFH41Nqz8KiNs8x0qy2e1QrNbmYFuZDUDkYSkHOT1JPuK7k0HhsG6u3SHPei
sLFwnucE95h0PdmC8DUNhR/OrCR5Ak9NvX9F6cRp20IZIR706AX1I5DA2Qn9KRsPvzJqGmXG0N6pf
k3KTEhW2ykBsLKUh2Y4nKlY6lKCAOuVmuafry5T5rFs0vaHQ5JSVIm3FBAQhVq7wfnWjurAPnQZ9p
mslWtlVmtDqRcnxwrdK+ERknqT0Vj7DftUN70AlBilyWwx65TnU8PjOatMsJJzlhbBypXMhPFfGADb
rF6B0sNSagkzrk+5cbTDeUQ5IHwy3T+bHbYE/y9yK9sabQ02lDSAhCRhKUjAAoIaFaZ8gh++zg8vI
KYsUFphB+/Ev8AmONuVTYGBgcqzSoFKUoFKUoFKUoFcV4mOW+2yJbEN6Y40gqTHYGVuHsK7awRmg8
rd/lgameJn2n8Ptv5YQn+78f+8cSFLI57AJ8/Oy2eyalhXkRo79htEVI2YgQ1UEH+JSgd68NW/FQ2
r7yqxWCTNzR4knZqM1/ePLPChP4i0KRC4l9lFqJ3TkFust2Aygfiz7TDTAEa8mkeAnjPXfAH2Noha
PehrZ2Y8bVF9QwygNttJMcJkSkDAAAZru0hY02CyMxFK8SWv9rMfPN5561E9d9vQCpughrLNZVx1l
Fc19kupYUN/wAYP+NdQVMjD+khEhSFm40nhYU2PmP3z9D9K76xQRN/uNxjWj3qWw9F0krKfDa8YISUn
83F25Vvocf2kXdr/EJlrskZQxiM14ryR5ZJA9c/SrHo973m2SFjPhCfKSzn9wPLCceXbyxUpPmRrd
DemTHUMx2UFbjizgJAoPGrbpyJBRPvEi53Stfhc34MRCVMrckLSvCT8aFEEjBUQdhUlfbHcrcItuY
vc+VqbUBDcpf7PgDsfNJPBxBCQcDBGcnlUn7Om4kly+aonhbBPBPkeGiRsIqCEqUChkSMZ9K5bRqqM
bjJlE5FkTrndFe72m2sAF1MVB14iD8iVKyoQ02w7VRbrJpRdkt7MGDe56GGhgJ8Njn1P8AZ9fPnd5
gXNKQExt0q7uRmz/0AQfJa29znGFq2EmxuqaLzDrkhK2XUj5gF7YUNvhqsX+66nvtqlarsrsmDa7a
UvQIQULuCUqHiLdGPK4c8I686g9BMK8/lu7IxyzCB/9q4bpJkWeP7xd9TQojGeHjdipRk9hlW5/y
rF2lraoEGM/HcM+TMSDEhwyFuv55YA5DfcnYVx2fTD9wm/jesKMS7goYyhY42IKeyQeajt1R68tqC
UjtxWVHakQL/FejupC23PcwsLSeoKVGEVsLWomsFMmlyAM5C2HGir6hSsfY1B6ILViuV50utSWkRp
HvNvQTgG060LCe/CviB7bVMXjVMC2vCG0VTrk4MtQIEHHleZHJCf1KwKDTcNSqsKb2XqSCyJtQ3fY
cDzSj0SOSgSdhlP1r50FqherbM5cVQVQwmQtKnlfFxBON8/XB8wa4JNsFUzJlJq/wXFQWnH4tvbPE
zFCUK8RyPjd2+bkOQ7nHs0iSLRY7dBKKUoy4gnYV8yHFEFFxPoCtOPU+VBdaUpQKUpQKqF4H4xry02
Y7YrYyq4yB3cPwND/vV9BVvNVTRf8ATrhqC9KIUJU4x2T2aZHAB/xcZ+tBa6UpQKr+r7lIjRWrdaz
/AFrcleBF2z4W3xOq8kDflwOtSV5uksZ2l6fOc4GGhk4GSo9EPHVROwHU1DaWtst2S9qG+N8Nzlp4
WWSc+6R85S0P1dVHqfSgm7Rb2bTbItvjcXhR2g2kqOVKx1J6k8zVbfp+leoSxkGx2h7LxztJlD8h6
FCOZ/VjtXdq25ymWY9ptSv61uSi2yr+4Rj43T5JHLzIrRfHIujtCy/dthFiqQyD8zrqhgZ7qUo/40
FJsbL2q7W/YychUdqdmLXC5SEDJQhqbq/CbHTKuEEj90edXfRWi4Gk4yvAUqTMdADsp35ikckj9lI7
Vn2e6bTpnTUA6G5hUtwelJcHVw9PQch6VZ6Dhudot12aQldIEWa2hXEhEl1LgSe4Cga7OBPBwYHDjG
MbYr6pQRNP03ZrM669arVChuu/OphkJJ8tunlUt0pSgjLxYLvew2LtAYleEctqcT8SPRXMfevq0W0
12VtTdppRoiVHKy02Avnuo8z9akaUEBR2O9L0beI8dtx1xyMpIQ2kqUodQANycZrk07JVeLyq4R2H
mrZCiiJFW62UF9Sils1AHfhHAgA7b8XlVqIzWMB0GaUpQKUpQc9wkCJAkyVcmWlOH6AmoL2btsqZ0L
Zi4SXHowfcUeZU58ZP3VUpqNlCjT1zYaGvURHUJA6koIFcuiZDcnR9lea+RcFkgdvgG1BNlomS48G
M7JluoZYasVuOLOEpSOZJrXdLjEtUF2bcJLceMOMrccOAP8AP0qqR4czWklqfd2HitgzWfXLe6MLl
KHJ14dE9kh1NBttDejVVzYvtxaUza4547ZCdThSz0kLHQ4+UdAc86tcmQzDjOyJLiW2WUFxxxWwSk
DJJ+lbQAOVVPU6vx29xdLsqPgBKZdzIG3ghWEtE9CtX/Kk0GzSTDlwekamNlL08BMNCs5ZijdaWe
RVniPqB0qsarcelXrezWlghVsiTCp3B/tfTdicPok8CP4lq7VedSzXYFr8OBwpmYVCPEyPhStQPxE
fupAKj5JNVz2eW9t2RivLJWqGlsQbetXNlpCsuPerjmvZ6gCqLyBis0pUclKUC1KUC1KUC1KUC1KU
ClKUGCMjFVNqw36yeOxpmbb/wAPdcU43GntLPuqlHJCFJO6ckkJI2zzq20oKtE0iZe5q46mnKu8to
8TLsmwiMwe6G99/wBSiTVoGwrNcV4uUez2yTcJiiGY7ZWRAyT2AHUK7DlOmXq6R7PapNxlk+FHQVk
AZKj0SB1JOAPWozRtqfhW5ybcf9qXJz3qZk54FEbNg9kDCfoT1qGi++alvEGJdGwlq2hE+e0FAPTK
VuOwe4Qk8R7nhNXkcqCs6q0zK1DcIWbkqNBw23ESmWk4ceCsZAV+UEAgngngvViix2okZqPHbS2y0
kIbQkYCUjYAVtpQKUpQKUpQKUpQKUpQKUpQKUpQKUpQKUpQKouv7mwi7W2HJbDZiJNXXHSd5DoUER
2gOpU4rI/gq9Vx02i3PXRu6OwmFz2m/DbkKQCTcKc4B+p+9BxaTtblqtQEvhM+UtUqatO4U8vdW/Y
bJHkkVNVGDFZofFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUo
FKUoFKUoP//Z"
}
]
} as CustomStampSettings);
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StampSettings, DynamicStampItem ,
SignStampItem, StandardBusinessStampItem, CustomStampSettings} from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addDynamicStamp()">Add Dynamic Stamp
Programmatically</button>
<button (click)="addSignStamp()">Add Sign Stamp Programmatically</button>
<button (click)="addStandardBusinessStamp()">Add StandardBusiness Stamp
Programmatically</button>
<button (click)="addCustomStamp()">Add Custom Stamp Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addDynamicStamp() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Stamp", {
      offset: { x: 200, y: 140 },
      pageNumber: 1
    } as StampSettings, DynamicStampItem.Approved);
  }
  addSignStamp() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Stamp", {
      offset: { x: 200, y: 240 },
      pageNumber: 1
    } as StampSettings, undefined, SignStampItem.Witness);
  }
  addStandardBusinessStamp() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];

```

```
pdfviewer.annotation.addAnnotation("Stamp", {
offset: { x: 200, y: 340 },
pageNumber: 1
} as StampSettings, undefined, undefined,
StandardBusinessStampItem.Approved);
}
addCustomStamp() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
viewer.annotation.addAnnotation('Stamp',
{
offset: { x: 100, y: 440 },
width: 46,
author: 'Guest',
height: 100,
isLock: true,
pageNumber: 1,
customStamps: [
{
customStampName: "Image",
customStampImageSource:
"data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAkGBwgHBGkIBwgKCgkLD
RYPDQwMDRsUFRAWIB0iIiAdHx8kKDQSJCyxXJ8fLT0tMTU3Ojo6Iys/RD84QzQ5OjcBCgoKDQwNGg
8PGJclHyU3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nzc3N//
AABEIAIIAQwMBIGACEQEDEQH/xAAAbAAEAAGMBAQAAAAAAAAAAAAABQYBawQHAV/EAEEQAAEDAwIE
AwYDBAYLAAAAAAAAECAwQAAREGIRIxQVETyXEhFCIygZEVMqIJUnKJCUCzU6HRFhc1c5KisbKzwvD/x
AAVAQEBAAAAAAAAAAAAAAAAAf/EABQRAQAAAAAAAAAAAAAAAAAAD/2gAMAwEAAhEDEQA/AP
caUpQKUppQKUppQKUppQKVzXGdFtsN2ZPFbyYNJ4nHHdGJffEK5Q5ttbuUaQhcNxvxUPcklPffflD
dlYJxURpe/salthuMNpxEYvuNtKc28VKVcPGB2JB577VyZ7pNuUxy26eWlCml8Mu4OI4kR/0oB2Wv
p2T17EJK43qDbloakOql7m6I7TanHVjOmHCQTjzxgVut89i4Mqdj8Y4VlC00IKFIUOYKTuOn0INRZ
ZtWkrVLuDpIIHHJLPK4nXldOJR5kk4A5DYDARVoWbHuVgTPjvF5Ul5xx5zhIBC4jkJyBlI+UHqEOF
jpSlApSlApSlApSlApSlApSlArClAczgVmqr7QZLptkezXhilKvD4ihxKsFprBU6v6IB+pFBT
dUKfluUuFa0WpyUIVoYBx706chchXdKEhZSPLNXXVTsOW6NdjNxxvJS0iLEidHnDhLaPME4z5ZzVH
k6kTHulvTpyElJf8L3OwwlZDaGc4XJXjklXDhp6ULWd63ybrLlrqlmNa1hLcAKEEQgcTbbvyuScHn
w5KGweZJPiVRyODT6okfslnfWhmCwl43FGAUkxu2j9atyT+UHvirZBixLZBaiQ2kR4zCMIQnZKRK
uz2yLZ7czBgo4GWhl3KidypR6qJJJPevOvaFqCXqC4HSGmzxlxQbmVJJAPXwwe2M8R9R3FQC1xde
qOqEW+C44jTFuVxpvtNHvcVi+e4HYZPavV4sdmLGajxmktMtJCENpGakDKBUbpixRNO2dm3Q0/Cj4
lrPNazzUf/uWKLkkeZByKDNKuOfKUOfKUOfKUOfKWahZ2p7dFFFMZhTs+ZnHuOFHirB/VjZHqogUE3
WOIYZUapZUlWBKUxLowQCFL/bv467DCEn6qr5i6btK5ht+ZOLXLcxllkiTxtr8whGG8fy0HdK1Fzo
rymhbjH8dpNLc+NY/lTk1XNTEoml2SCXBfnrkOpX7uh6ItkKUBzPGEnhzcAlbokKLASlBjMx20j
AQy2EtHsK85i6PuOovaFNvpWPdrt8J/ggMKUCXktq+Bx8HNxmT2G9HLFltrSyW2GrUFGbluT3eCjr
IS26tS/iSJwgCcDl35Z3qBLsb/edVcn58e4toJkaFiW2MFdQpRBXF8X5UnZPGsq5ZAroTV2j52oL9
Anx7wqCxHYWypLbeXAFH4lnqz8KiNs8x0qy2elQrNbmyFuZDUdkYSKHOT1JPuk7k0HhsG6u3SHPei
sLFwnuce95hoPdmc8DUNhR/orCR5Ak9NvX9F6cRp20IZIR706AXlI5DA2Qn9KRSPvzJqGmXGON6pf
k3KTEhW2ykBsLKUh2Y4nKly6lKCAOuVmuafr5Y5TrFs0vaHQ5JSVIm3FBaQhvq7wfNwjurAPnQZ9p
mslwtlVmtDqRcnxwrDK+ERknqT0Vj7DftUN7OA1BilyWwx65TnU8PjOAtMsJJzlhbByPMhPfGADB
rF6B0sNSagkzrk+5cbTDeUQ5IHwy3T+bHbYE/y9yK9sabQ02lDSAhCrHKUJAaoIaFaZ8gh++zgsvI
KYsUFphB+/Ev8AmONuVTYGBgcqzSoFKUOfKUOfKUOfFcV4mOW+2yJbEN6Y40gqTHYGvUHSK7awRmg8
rd/lgameJn2n8PtV5Yqn+78f+8csFLl57AJ8/Oy2eyalhxrRo79htEVI2YgQluEH+JSgD68NW/FQ2
r7yqxWCTNZR4knZqm1/ePLPChP3IoKRc4l9lfqJ3TkfUst2Aygfiz7TDTaEA8mkEANjPxFAH2Noha
PehR2Y8bvF9QwygnTtJMcJSKDAAAZru0hy02CyMxFK8SWv9rMfPN55W6le9d9vQCpughmrLNZVxIl
Fc19kupYun/wAYP+NdQMJD+khEhsfm40nhUPMp3z9D9K76xQRN/unXjWj3qwW9F0krKfda8YISUn
83F25VVocf2kXdr/EJlrskZQxiMl4ryR5ZJA9c/SrHo973m2SFjPhCFkszn9wPLCCexbyxUpPmRrd
DemTHUmX2UFbjizgJAOPgrbpyJBRPVEi53STfhc34MRVCmrckLSvCT8aFEeJBUQdhU1fbHcrctYuY
vc+vqbUBDcpf7PgDsfnJPBXBCQcDBGcnlUn7Om4kly+aonhbPBfpkeGiRSIqCEqUChKSzMZ9K5brqqM
bjJ1E5fkTrndFe72mSAFIMVBI4id8iVKyoq2w7VRBrJpRdkt7MGDe56GGhgJ8Njnlp8AZ9sfPnd5
gXNKQEXT0q7uRmz/0AQfJa29znGFq2EmxuqaLDrkhK2XUj5gF7YUNvhqsX+66nvtqlarsrmsDa7a
UvQIqQUlLuCUgHiLdGPk4c8I686g9BMK8/lu7IxyzCB/9q4bpJkWeP7xd9TQojGeHjdipRk9hlW5/y
```



```

rF21raoEGM/HcM+TMSDEhwyFuv55YA5DfcYVx2fTD9wm/jeskMS7goYYhY42IKeyQeajtlR68tqC
UjtXWVHakQL/FejupC23PcwsLSeoKVgEVsLWomsFMmlyAM5C2HGir6hSsfY1B6ILViuV50utSWkRp
HvNvQTgGO6OLCe/CviB7bVMXjVMC2vCG0VTrk4MtQIEHHleZHJCf1KwKDTcNSqskB2XqSCYjTQ3fY
cDzSj0SOSgSdh1Plr50FqherbM5cVQVQwmQtKnlfFxBON8/XB8wa4JNsfUzJ1Jq/wXFQWnH4tvpPE
zFCUk8RyPjd2+bkOQ7nHs0iSLRY7dBkKUoy4gnYV8yHFEFxFPoCtOPU+VBdaUpQKUpQKqF4H4xry02
7YxrYyq4yB3cPwND/vV9BVvNVTRf8ATrhqC9KIUJU4x2T2aZHAB/xcZ+tBa6UpQKr+r7lIjRWrdaz
/AFrcleBF2z4W3xOq8kDf1wOtSV5ukSz216fOc4GGhk4GSo9EpHVROwHU1DaWtst2S9qG+N8Nzlp4
WWSc+6R85S0PlDVHqfSgm7Rb2bTbItvjcXhR2g2kqOVKx1J6k8zVbfp+leoSxkGx2h7LxztJ1D8h6
FCOZ/VjtXdq25ymWY9ptSv61uSi2yr+4Rj43T5JHLzIrRfHIujtCy/dthFiqQyD8zrqhgZ7qUo/40
FJsbL2q7W/YYchUdqMlXC5SEDJQhbq/CbHTKuEEj90edXfRWi4Gk4yvAUqTMdAdsp35ikckj91I7
Vn2e6bTpnTUaG5hUtweLJCHVw9PQch6VZ6Dhudot12aQ1dIEWa2hXEhEllLgSe4Cga7OBPBWYHDJg
MbYr6pQRNp03ZrM669arVChuu/OphkJJ8tunlUt0pSgjLxYLVew2LtAYleEctqct8SPRXMfevq0W0
12VtTdpqRoiVHKY02AVnuo8z9akaUEBr2O9L0beI8dtx1xyMpIQ2kqUodQANycZrk07JVeLyq4R2H
mrZCiiJFW62UF9SilS1AHfhHAgA7b8X1VqIzWMb0GaUpQKUpQc9wkCJakyVcmW1OH6AmoL2bsqZ0L
Zi4SXHowfcUeZU58ZP3VUpqNlcjT1zYaGVuRHUJA6koIFcuiZDcnR9lea+RcFkgdvgG1BN1omS48G
M7JluoZYaSVuOLOEpSOZJrXdLjEtUF2bcJLceM0MrccOAP8AP0qqR4czWk1qfd2HitgZWfXLe6ML1
KHJ14dE9kH1NBttDEjVVzYvtxaUza4547ZCdThSz0kLHQ4+UdAc86tcmQzDjOyJLiW2WUFxxxWwSk
DJJ+1bQAOVVPu6vx29xdLsqPgBKZdzIG3ghWEtE9CtX/Kk0GzSTDlwekamnN1L08BMNCs5ZijdAwe
RVniPqB0qsarcelXrezWlghVsiTCp3B/tFtDicPok8CP4lq7VedSzXYFr8OBwpmyVCPEyPhStQPxE
fupAKj5JNVz2eW9t2RivLJWqGlsQbetXN1pCsuPerjmVZ6gCqLyBis0pUClKUC1KUC1KUC1KUC1KU
ClKUGCMjFVNqW36yeOxpmBB/wAPdcU43GntLPuqlHJCFJO6ckkJI2zzq20oKtE0iZE5q46mnKu8to
8TlSmwiMwe6G99/wBSiTvoGwrNcV4uUez2yTcJiiGY7ZWRAyT2AHUk7D1oMXq6R7PapNxlk+FHQV
AZKj0SB1JOAPWozRtqfhw5ybcf9qXJz3qZk54FEbNg9kDCfoT1qGi++alvEGJdGwlq2hE+e0FApTK
Vu0we4Qk8R7nhNXkcqCs6q0zK1DcIWbkqNbW23ESmWk4ceCsZAV+UEAgngngvViix2okZqPhbS2y0
kIbQkYCUjYAVtpQKUpQKUpQKUpQKUpQKUpQKUpQKUpQKUpQKUpQKouv7mwi7W2HJBdZiJNXXHsd5DoUER
2gOpU4rI/gq9Vx02i3PXRu6OwmFz2m/DbkKQcTck4B+p+9BxaTtblqtQEvhM+UtUqatO4U8vdW/Y
bJHkkVNVgDFZofKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUoFKUo
FKUoFKUoP//Z"
}
]
} as CustomStampSettings);
}
}
}

```

Edit the existing sticky note annotation programmatically

To modify existing sticky note annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method:

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editAnnotation()">Edit Stamp annotation</button>
<ejs-pdfviewer
id="pdfViewer"

```



```

[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
editAnnotation() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++)
{
if (viewer.annotationCollection[i].shapeAnnotationType === "stamp") {
var width = viewer.annotationCollection[i].bounds.width;
var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotationCollection[i].annotationSettings.isLock = true;
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editAnnotation()">Edit Stamp annotation</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,

```

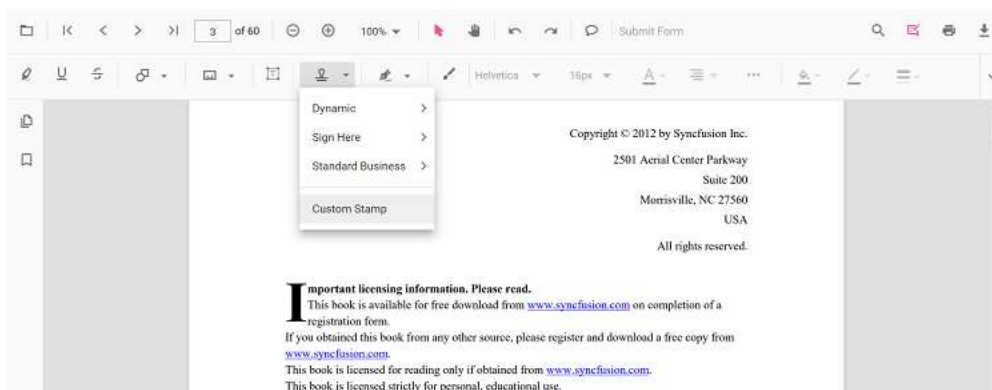
```

providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
editAnnotation() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++)
{
if (viewer.annotationCollection[i].shapeAnnotationType === "stamp") {
var width = viewer.annotationCollection[i].bounds.width;
var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotationCollection[i].annotationSettings.isLock = true;
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}
}

```

Adding custom stamp to the PDF document

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Click the **Stamp Annotation** drop-down button. A drop-down pop-up will appear and shows the stamp annotations to be added.
- Click the Custom Stamp button.



- The file explorer dialog will appear, choose the image and then add the image to the PDF page.

The JPG and JPEG image format is only supported in the custom stamp annotations.

Setting default properties during control initialization

The properties of the stamp annotation can be set before creating the control using the StampSettings.

After editing the default opacity using the Edit Opacity tool, they will be changed to the selected values.

Refer to the following code sample to set the default sticky note annotation settings.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[stampSettings]='stampSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public stampSettings = { opacity: 0.3, author: 'Guest User' };
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[stampSettings]='stampSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public stampSettings = { opacity: 0.3, author: 'Guest User' };
}
```

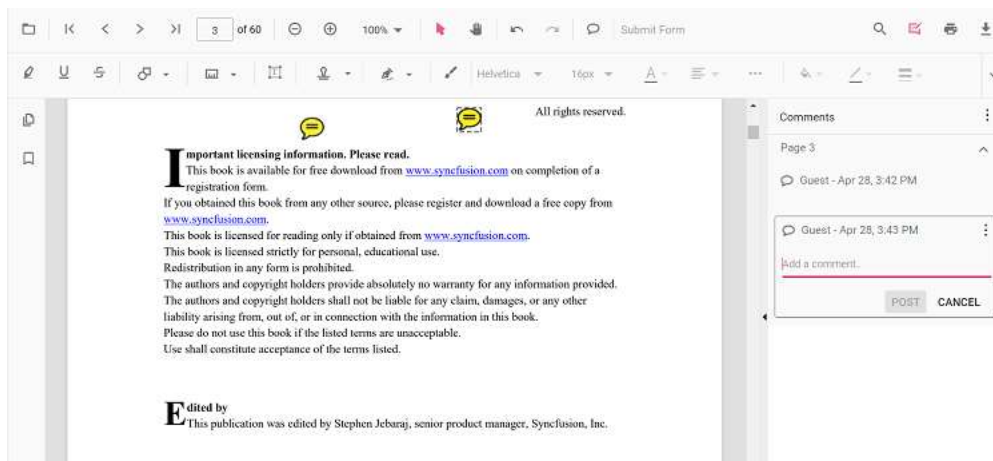
```

providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public stampSettings = { opacity: 0.3, author: 'Guest User' };
}

```

Sticky Notes Annotation in the Angular PDF Viewer component

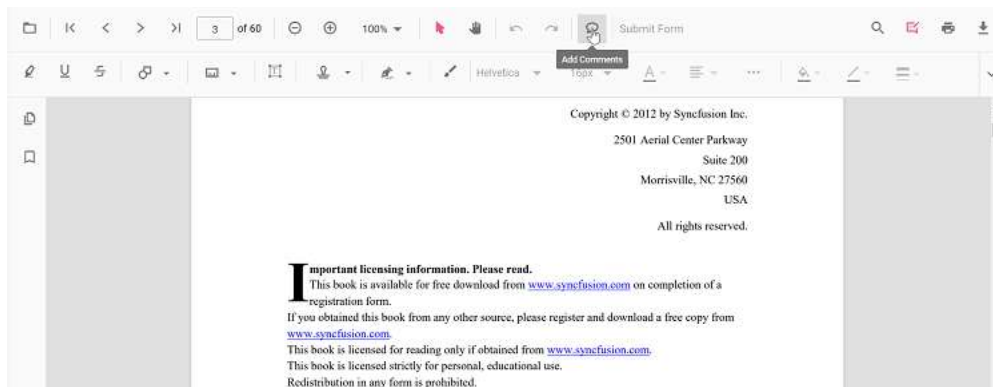
The PDF Viewer control provides the options to add, edit, and delete the sticky note annotations in the PDF document.



Adding a sticky note annotation to the PDF document

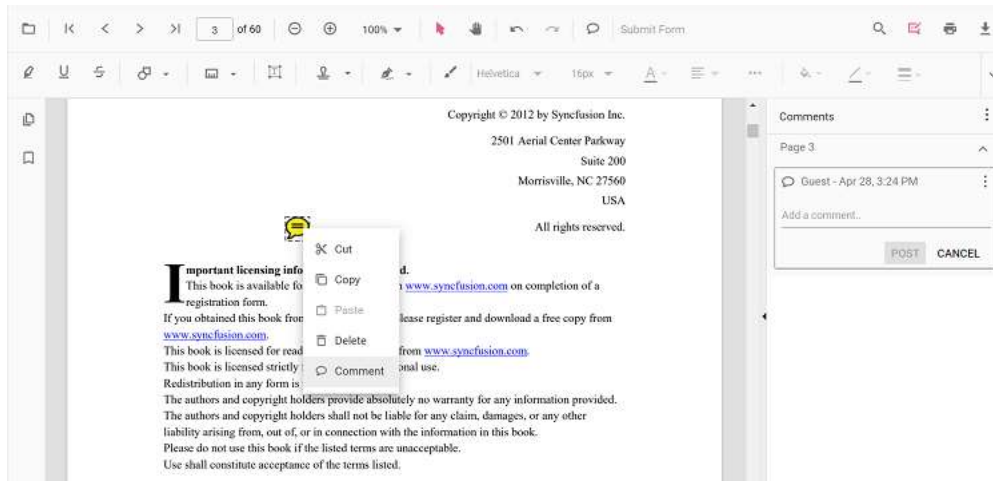
Sticky note annotations can be added to the PDF document using the annotation toolbar.

- Click the **Comments** button in the PDF Viewer toolbar. A toolbar appears below it.
- Click the position where you want to add sticky note annotation in the PDF document.
- Sticky note annotation will be added in the clicked positions.



Annotation comments can be added to the PDF document using the comment panel.

- Select a Sticky note annotation in the PDF document and right-click it.
- Select the Comment option in the context menu that appears.
- Now, you can add Comments, Reply, and Status using the Comment Panel.
- Now, you can add Comments, Reply, and Status using the Comment Panel.



Adding a sticky note annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a sticky note annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's a example of how you can utilize the **addAnnotation()** method to include a sticky note annotation programmatically:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StickyNotesSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add StickyNotes Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
```

```

AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("StickyNotes", {
offset: { x: 100, y: 200 },
pageNumber: 1,
isLock: false
} as StickyNotesSettings);
}
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, StickyNotesSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add StickyNotes Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {

```

```
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("StickyNotes", {
  offset: { x: 100, y: 200 },
  pageNumber: 1,
  isLock: false
} as StickyNotesSettings);
}
```

Edit the existing sticky note annotation programmatically

To modify existing sticky note annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="editAnnotation()">edit sticky note</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  editAnnotation() {
    var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    for (let i = 0; i < viewer.annotationCollection.length; i++)
    {
      if (viewer.annotationCollection[i].shapeAnnotationType === "sticky") {
        var width = viewer.annotationCollection[i].bounds.width;
```

```

var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editAnnotation()">edit sticky note</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
editAnnotation() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++)
{
if (viewer.annotationCollection[i].shapeAnnotationType === "sticky") {
var width = viewer.annotationCollection[i].bounds.width;
var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}

```

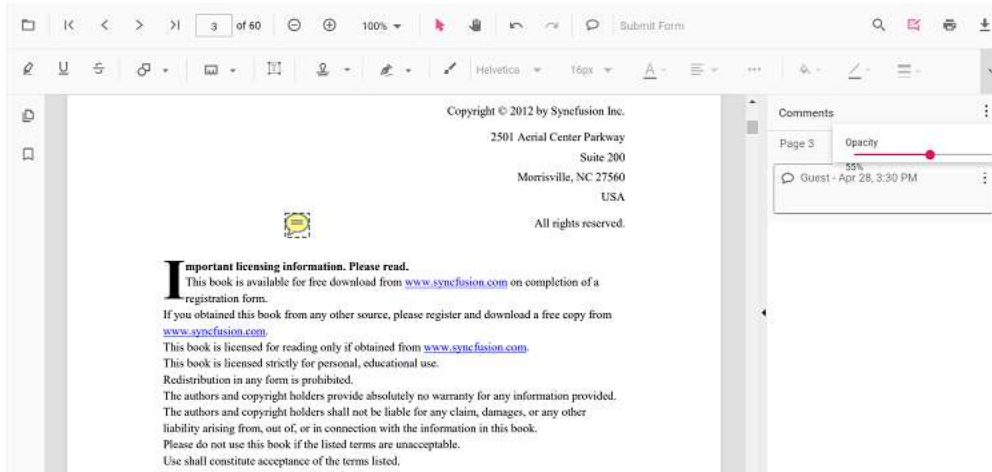


```
}
}
```

Editing the properties of the sticky note annotation

Editing opacity

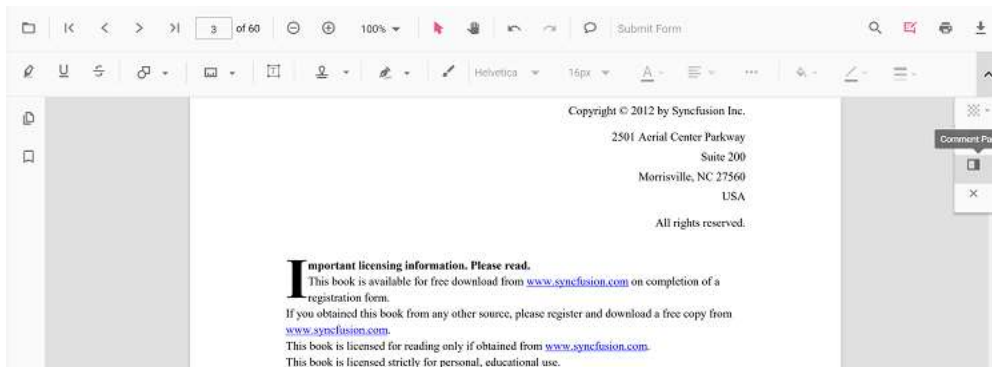
The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



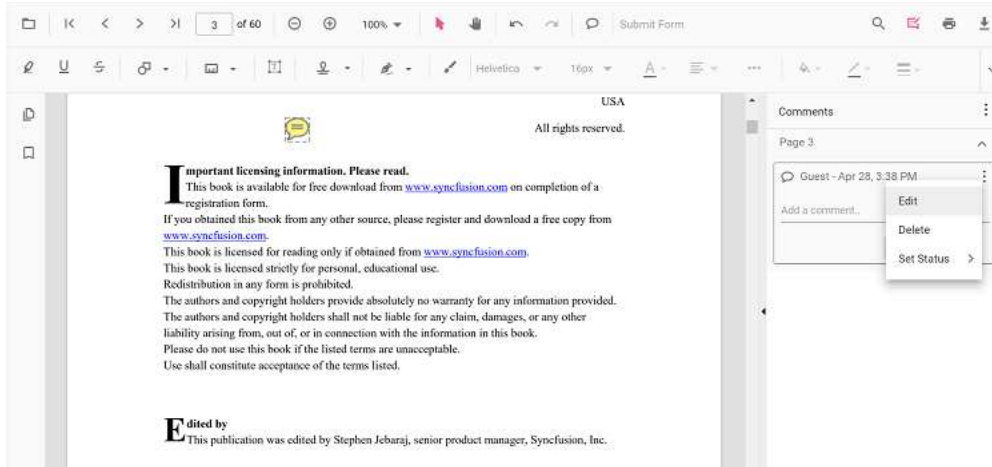
Editing comments

The comment, comment reply, and comment status of the annotation can be edited using the Comment Panel.

- Open the comment panel using the Comment Panel button showing in the annotation toolbar.



You can modify or delete the comments or comments replay and it's status using the menu option provided in the comment panel.



Setting default properties during the control initialization

The properties of the sticky note annotation can be set before creating the control using the `StickyNoteSettings`.

After editing the default opacity using the Edit Opacity tool, they will be changed to the selected values. Refer to the following code sample to set the default sticky note annotation settings.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]='document'
  [resourceUrl]='resource'
  [stickyNotesSettings]='stickyNotesSettings'
  style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  public stickyNotesSettings = { author: 'Syncfusion' };
```

```
ngOnInit(): void {
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
[stickyNotesSettings]='stickyNotesSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public stickyNotesSettings = { author: 'Syncfusion' };
ngOnInit(): void {
}
}
```

Disabling sticky note annotations

The PDF Viewer control provides an option to disable the sticky note annotations feature. The code sample for disabling the feature is as follows.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
```

```

@Component ({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer
      id="pdfViewer"
      [documentPath]='document'
      [resourceUrl]='resource'
      [enableStickyNotesAnnotation]='false'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService,
    PageOrganizerService]
  })
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
    pdfviewer-lib";
  ngOnInit(): void {
  }
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
  MagnificationService, ThumbnailViewService, ToolbarService,
  NavigationService, TextSearchService, TextSelectionService,
  PrintService, FormDesignerService, FormFieldsService,
  AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
  pdfviewer';
@Component ({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
    <ejs-pdfviewer
      id="pdfViewer"
      [documentPath]='document'
      [serviceUrl]='service'
      [enableStickyNotesAnnotation]='false'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
    MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService,
    PageOrganizerService]
  })

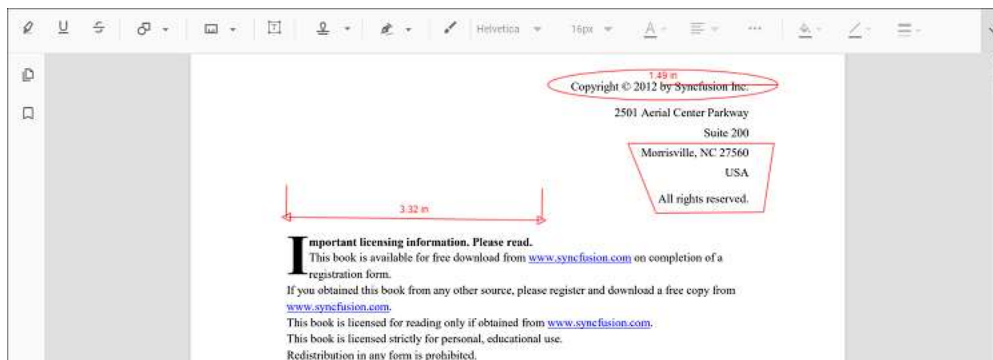
```

```
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
  public service: string =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
}
```

Measurement Annotation in Angular PDF Viewer component

The PDF Viewer provides the options to add measurement annotations. You can measure the page annotations with the help of measurement annotation. The supported measurement annotations in the PDF Viewer control are:

- Distance
- Perimeter
- Area
- Radius
- Volume

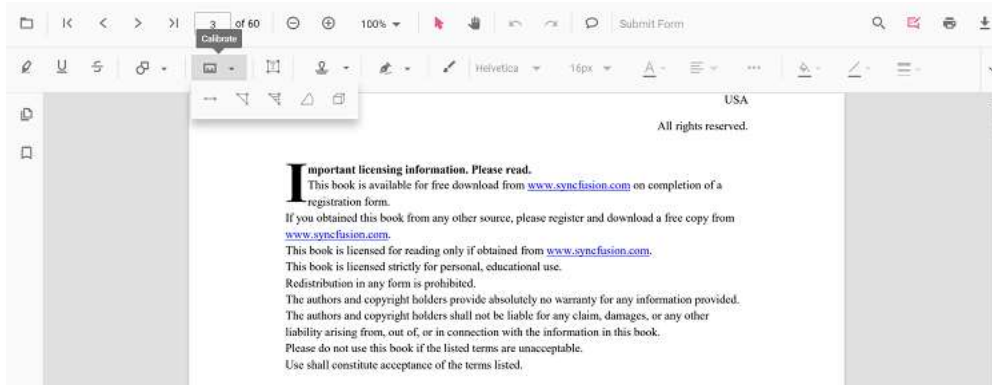


Adding measurement annotations to the PDF document

The measurement annotations can be added to the PDF document using the annotation toolbar.

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Click the **Measurement Annotation** dropdown button. A dropdown pop-up will appear and shows the measurement annotations to be added.
- Select the measurement type to be added to the page in the dropdown pop-up. It enables the selected measurement annotation mode.
- You can measure and add the annotation over the pages of the PDF document.

In the pan mode, if the measurement annotation mode is entered, the PDF Viewer control will switch to text select mode.



Refer to the following code snippet to switch to distance annotation mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Distance</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode('Distance');
}
}
```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Distance</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode('Distance');
  }
}

```

Adding a measurement annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a measurement annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's a example of how you can utilize the **addAnnotation()** method to include a measurement annotation programmatically:

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, DistanceSettings, PerimeterSettings,

```

```

AreaSettings, RadiusSettings, VolumeSettings } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addDistanceAnnotation()">Add Distance annotation
Programmatically</button>
<button (click)="addPerimeterAnnotation()">Add Perimeter annotation
Programmatically</button>
<button (click)="addAreaAnnotation()">Add Area annotation
Programmatically</button>
<button (click)="addRadiusAnnotation()">Add Radius annotation
Programmatically</button>
<button (click)="addVolumeAnnotation()">Add Volume annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addDistanceAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Distance", {
      offset: { x: 200, y: 230 },
      pageNumber: 1,
      vertexPoints: [{ x: 200, y: 230 }, { x: 350, y: 230 }]
    } as DistanceSettings);
  }
  addPerimeterAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Perimeter", {
      offset: { x: 200, y: 350 },
      pageNumber: 1,
      vertexPoints: [{ x: 200, y: 350 }, { x: 285, y: 350 }, { x: 286, y: 412 }]
    } as PerimeterSettings);
  }
  addAreaAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("Area", {
      offset: { x: 200, y: 500 },

```



```

pageNumber: 1,
vertexPoints: [{ x: 200, y: 500 }, { x: 288, y: 499 }, { x: 289, y: 553 }, {
x: 200, y: 500 }]
} as AreaSettings);
}
addRadiusAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Radius", {
offset: { x: 200, y: 630 },
pageNumber: 1,
width: 90,
height: 90
} as RadiusSettings);
}
addVolumeAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Volume", {
offset: { x: 200, y: 810 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 810 }, { x: 200, y: 919 }, { x: 320, y: 919 }, {
x: 320, y: 809 }, { x: 200, y: 810 }]
} as VolumeSettings);
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, DistanceSettings, PerimeterSettings,
AreaSettings, RadiusSettings, VolumeSettings } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addDistanceAnnotation()">Add Distance annotation
Programmatically</button>
<button (click)="addPerimeterAnnotation()">Add Perimeter annotation
Programmatically</button>
<button (click)="addAreaAnnotation()">Add Area annotation
Programmatically</button>
<button (click)="addRadiusAnnotation()">Add Radius annotation
Programmatically</button>
<button (click)="addVolumeAnnotation()">Add Volume annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,

```

```

providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addDistanceAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Distance", {
offset: { x: 200, y: 230 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 230 }, { x: 350, y: 230 }]
} as DistanceSettings);
}
addPerimeterAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Perimeter", {
offset: { x: 200, y: 350 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 350 }, { x: 285, y: 350 }, { x: 286, y: 412 }]
} as PerimeterSettings);
}
addAreaAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Area", {
offset: { x: 200, y: 500 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 500 }, { x: 288, y: 499 }, { x: 289, y: 553 }, {
x: 200, y: 500 }]
} as AreaSettings);
}
addRadiusAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Radius", {
offset: { x: 200, y: 630 },
pageNumber: 1,
width: 90,
height: 90
} as RadiusSettings);
}
addVolumeAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Volume", {
offset: { x: 200, y: 810 },
pageNumber: 1,
vertexPoints: [{ x: 200, y: 810 }, { x: 200, y: 919 }, { x: 320, y: 919 }, {
x: 320, y: 809 }, { x: 200, y: 810 }]
} as VolumeSettings);
}
}

```

```
}

```

Edit the existing measurement annotation programmatically

To modify existing measurement annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method:

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="editDistanceAnnotation()">Edit Distance annotation
Programmatically</button>
<button (click)="editPerimeterAnnotation()">Edit Perimeter annotation
Programmatically</button>
<button (click)="editAreaAnnotation()">Edit Area annotation
Programmatically</button>
<button (click)="editRadiusAnnotation()">Edit Radius annotation
Programmatically</button>
<button (click)="editVolumeAnnotation()">Edit Volume annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  editDistanceAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
      if (pdfviewer.annotationCollection[i].subject === "Distance calculation") {
```

```

pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editPerimeterAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Perimeter calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editAreaAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Area calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editRadiusAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Radius calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editVolumeAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Volume calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";

```

```
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editDistanceAnnotation()">Edit Distance annotation
Programmatically</button>
<button (click)="editPerimeterAnnotation()">Edit Perimeter annotation
Programmatically</button>
<button (click)="editAreaAnnotation()">Edit Area annotation
Programmatically</button>
<button (click)="editRadiusAnnotation()">Edit Radius annotation
Programmatically</button>
<button (click)="editVolumeAnnotation()">Edit Volume annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
editDistanceAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Distance calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
}
```

```

pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editPerimeterAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Perimeter calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editAreaAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Area calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editRadiusAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Radius calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}
editVolumeAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < pdfviewer.annotationCollection.length; i++) {
if (pdfviewer.annotationCollection[i].subject === "Volume calculation") {
pdfviewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
pdfviewer.annotationCollection[i].strokeColor = "#0000FF";
pdfviewer.annotationCollection[i].thickness = 2 ;
pdfviewer.annotationCollection[i].fillColor = "#FFFF00";
pdfviewer.annotation.editAnnotation(pdfviewer.annotationCollection[i]);
}
}
}

```

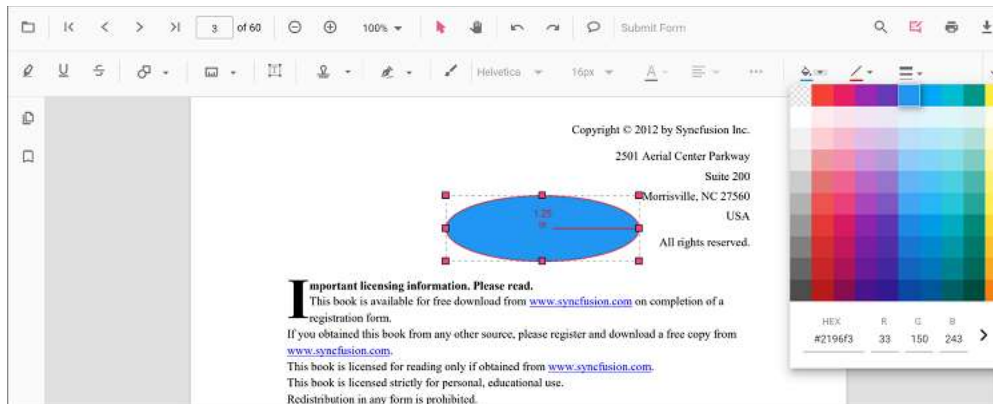
```
}  
}  
}
```

Editing the properties of measurement annotation

The fill color, stroke color, thickness, and opacity of the measurement annotation can be edited using the Edit Color tool, Edit Stroke Color tool, Edit Thickness tool, and Edit Opacity tool in the annotation toolbar.

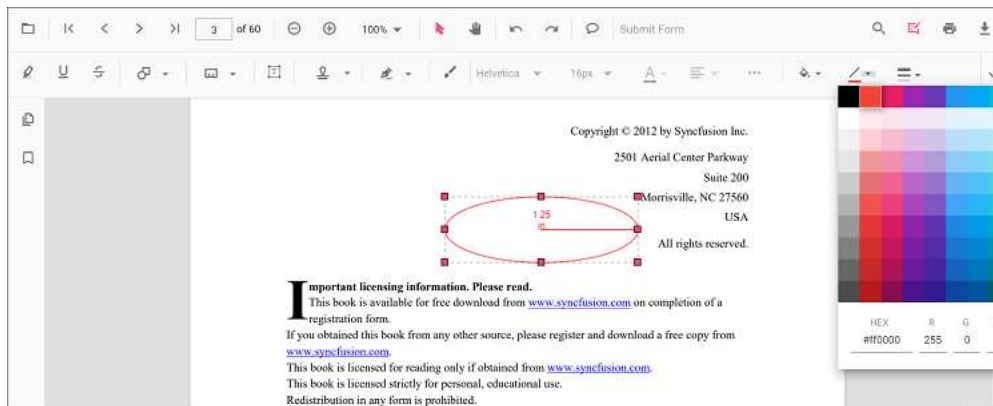
Editing fill color

The fill color of the annotation can be edited using the color palette provided in the Edit Color tool.



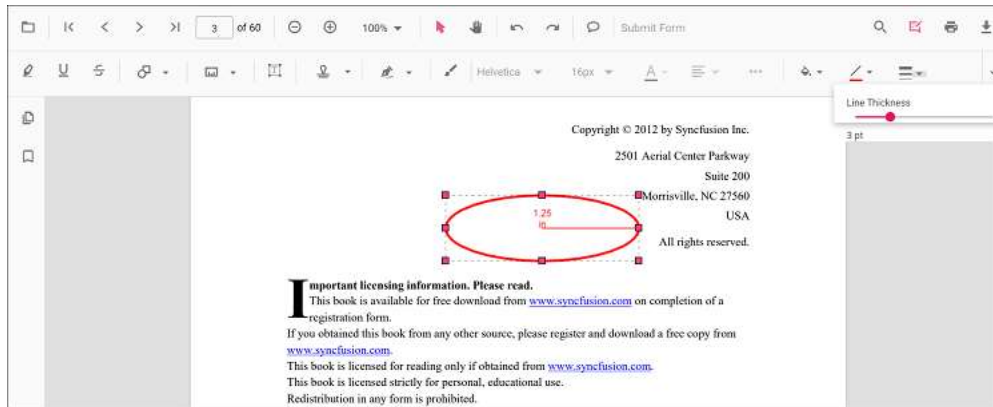
Editing stroke color

The stroke color of the annotation can be edited using the color palette provided in the Edit Stroke Color tool.



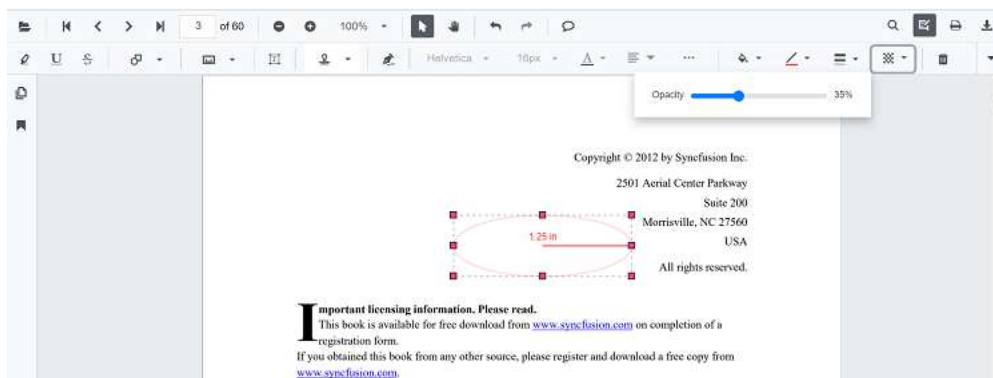
Editing thickness

The thickness of the border of the annotation can be edited using the range slider provided in the Edit thickness tool.



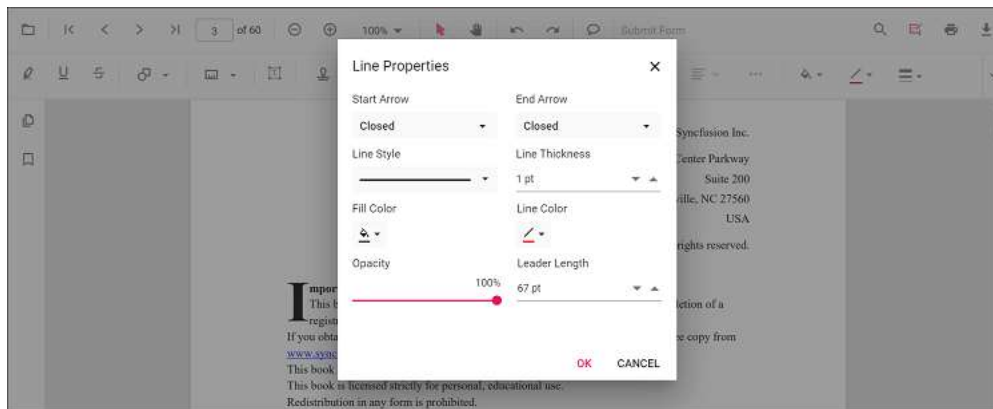
Editing opacity

The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



Editing the line properties

The properties of the line measurements such as distance and perimeter annotations can be edited using the Line properties window. It can be opened by selecting the Properties option in the context menu that appears on right-clicking the distance and perimeter annotations.



Setting default properties during control initialization

The properties of the measurement annotations can be set before creating the control using `distanceSettings`, `perimeterSettings`, `areaSettings`, `radiusSettings` and `volumeSettings`.

Refer to the following code snippet to set the default annotation settings.

STANDALONE


```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
[distanceSettings]='distanceSettings'
[perimeterSettings]='perimeterSettings'
[areaSettings]='areaSettings'
[radiusSettings]='radiusSettings'
[volumeSettings]='volumeSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
public distanceSettings = { fillColor: 'blue', opacity: 0.6, strokeColor:
'green' };
public perimeterSettings = { fillColor: 'green', opacity: 0.6, strokeColor:
'blue' };
public areaSettings = { fillColor: 'yellow', opacity: 0.6, strokeColor:
'orange' };
public radiusSettings = { fillColor: 'orange', opacity: 0.6, strokeColor:
'pink' };
public volumeSettings = { fillColor: 'pink', opacity: 0.6, strokeColor:
'yellow' };
ngOnInit(): void {
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,

```

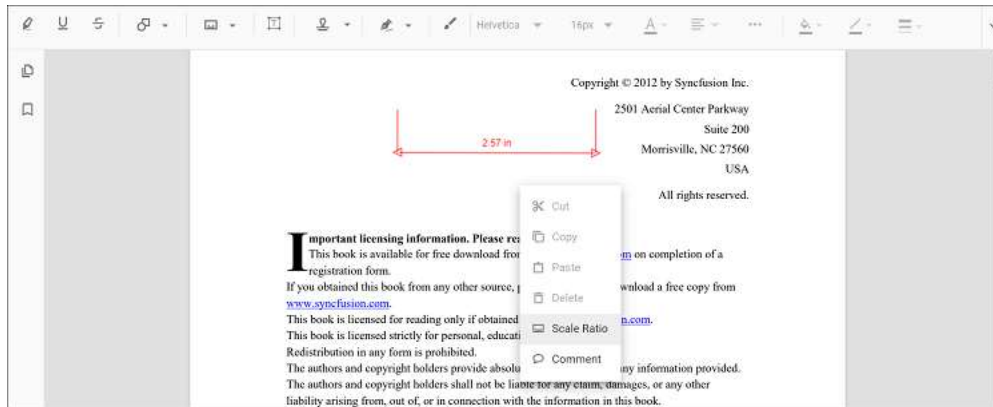
```

AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
[distanceSettings]='distanceSettings'
[perimeterSettings]='perimeterSettings'
[areaSettings]='areaSettings'
[radiusSettings]='radiusSettings'
[volumeSettings]='volumeSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public distanceSettings = { fillColor: 'blue', opacity: 0.6, strokeColor:
'green' };
  public perimeterSettings = { fillColor: 'green', opacity: 0.6, strokeColor:
'blue' };
  public areaSettings = { fillColor: 'yellow', opacity: 0.6, strokeColor:
'orange' };
  public radiusSettings = { fillColor: 'orange', opacity: 0.6, strokeColor:
'pink' };
  public volumeSettings = { fillColor: 'pink', opacity: 0.6, strokeColor:
'yellow' };
  ngOnInit(): void {
  }
}

```

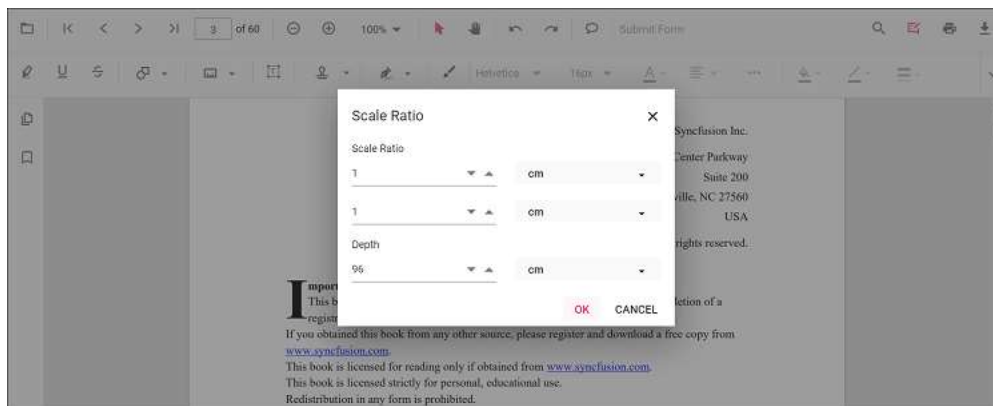
Editing scale ratio and unit of the measurement annotation

The scale ratio and unit of measurement can be modified using the scale ratio option provided in the context menu of the PDF Viewer control.



The Units of measurements support for the measurement annotations in the PDF Viewer are

- Inch
- Millimeter
- Centimeter
- Point
- Pica
- Feet



Setting default scale ratio settings during control initialization

The properties of scale ratio for measurement annotation can be set before creating the control using `ScaleRatioSettings` as shown in the following code snippet,

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer`
```

```

id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
[measurementSettings]='measurementSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
public measurementSettings = { scaleRatio: 2, conversionUnit: 'cm',
displayUnit: 'cm' };
ngOnInit(): void {
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
[measurementSettings]='measurementSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';

```

```

public service: string =
  'https://services.syncfusion.com/angular/production/api/pdfviewer';
public measurementSettings = { scaleRatio: 2, conversionUnit: 'cm',
  displayUnit: 'cm' };
ngOnInit(): void {
}
}

```

Free text annotation

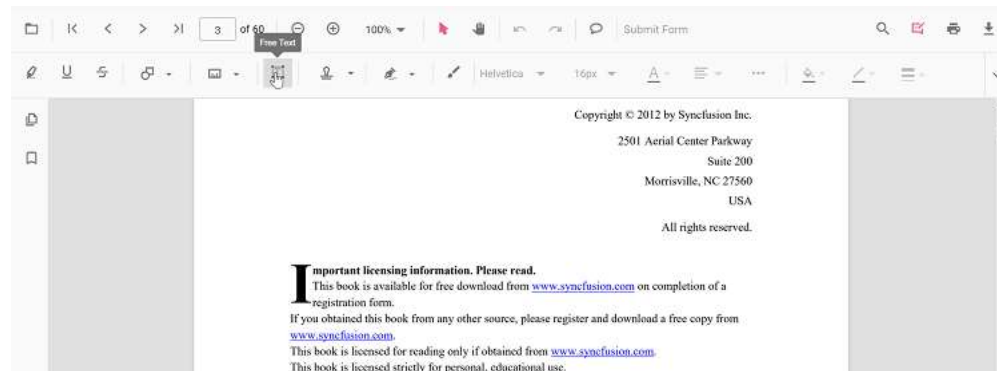
The PDF Viewer control provides the options to add, edit, and delete the free text annotations.

Adding a free text annotation to the PDF document

The Free text annotations can be added to the PDF document using the annotation toolbar.

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Free Text Annotation** button in the annotation toolbar. It enables the Free Text annotation mode.
- You can add the text over the pages of the PDF document.

In the pan mode, if the free text annotation mode is entered, the PDF Viewer control will switch to text select mode.



Refer to the following code sample to switch to the Free Text annotation mode using a button click.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService,
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<button (click)="addAnnotation()">Add FreeText annotation</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl] = 'resource'
style="height:640px;display:block">

```

```

</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService, FormDesignerService,
FormFieldsService, AnnotationService, PageOrganizerService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("FreeText");
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService,
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// Specifies the template string for the PDF Viewer component.
template: `<button (click)="addAnnotation()">Add FreeText annotation</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[serviceUrl] = 'service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService, FormDesignerService,
FormFieldsService, AnnotationService, PageOrganizerService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];

```

```
pdfviewer.annotationModule.setAnnotationMode("FreeText");
}
}
```

How to clear the selection focus from free text annotation

The free text annotations selection focus can be cleared by using the `setAnnotationMode` property of the `annotationModule`.

Refer to the following code sample to remove the selection focus from the annotation by using a button click.

STANDALONE

```
<button (click)="RemoveSelection()">RemoveSelection</button>
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
//Event triggers while clicking the RemoveSelection button.
RemoveSelection() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
//API to remove the selection from the free text annotation.
pdfviewer.annotationModule.setAnnotationMode('None');
}
```

SERVER-BACKED

```
<button (click)="RemoveSelection()">RemoveSelection</button>
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
//Event triggers while clicking the RemoveSelection button.
RemoveSelection() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
//API to remove the selection from the free text annotation.
pdfviewer.annotationModule.setAnnotationMode('None');
}
```

[View sample in GitHub](#)

Adding a Free Text annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a Free Text annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's an example of how you can utilize the `addAnnotation()` method to include a Free Text annotation programmatically

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
```

```

PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, FreeTextSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add FreeText annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("FreeText", {
      offset: { x: 100, y: 150 },
      fontSize: 16,
      fontFamily: "Helvetica",
      pageNumber: 1,
      width: 200,
      height: 40,
      isLock: false,
      textAlignment : 'Center',
      borderStyle : 'solid',
      borderWidth : 2,
      borderColor : 'red',
      fillColor : 'blue',
      fontColor: 'white',
      defaultText: "Syncfusion"
    } as FreeTextSettings);
  }
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,

```



```

PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, FreeTextSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add FreeText annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.annotation.addAnnotation("FreeText", {
      offset: { x: 100, y: 150 },
      fontSize: 16,
      fontFamily: "Helvetica",
      pageNumber: 1,
      width: 200,
      height: 40,
      isLock: false,
      textAlignment : 'Center',
      borderStyle : 'solid',
      borderWidth : 2,
      borderColor : 'red',
      fillColor : 'blue',
      fontColor: 'white',
      defaultText: "Syncfusion"
    } as FreeTextSettings);
  }
}

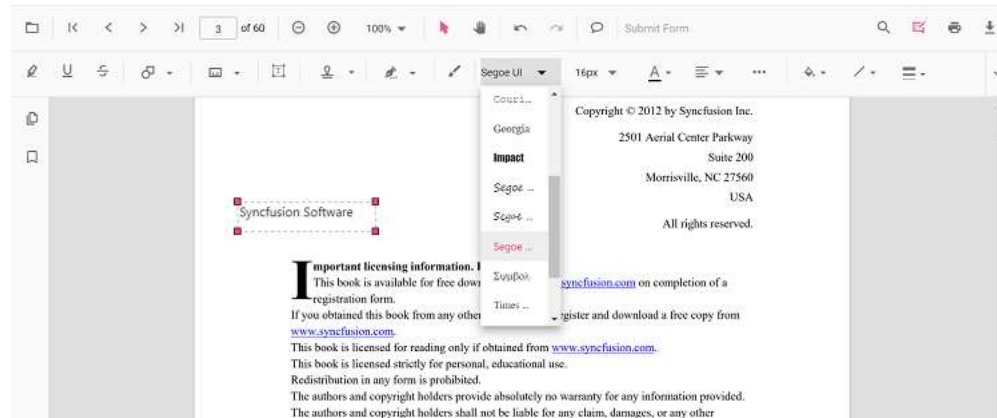
```

Editing the properties of free text annotation

The font family, font size, font styles, font color, text alignment, fill color, the border stroke color, border thickness, and opacity of the free text annotation can be edited using the Font Family tool, Font Size tool, Font Color tool, Text Align tool, Font Style tool, Edit Color tool, Edit Stroke Color tool, Edit Thickness tool, and Edit Opacity tool in the annotation toolbar.

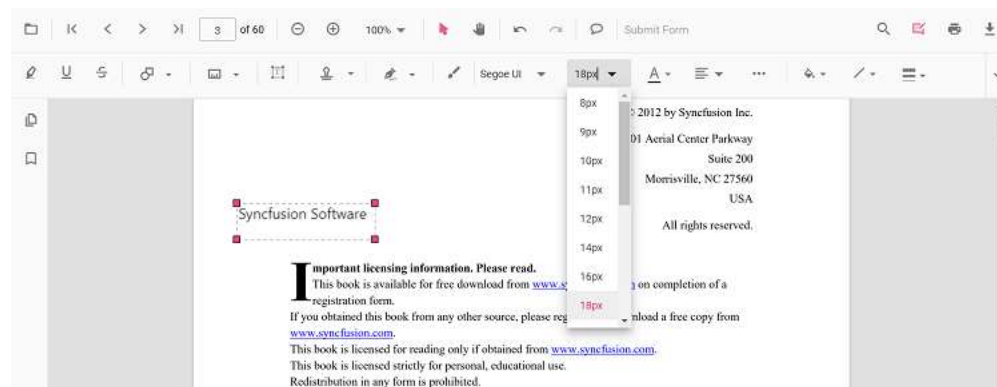
Editing font family

The font family of the annotation can be edited by selecting the desired font in the Font Family tool.



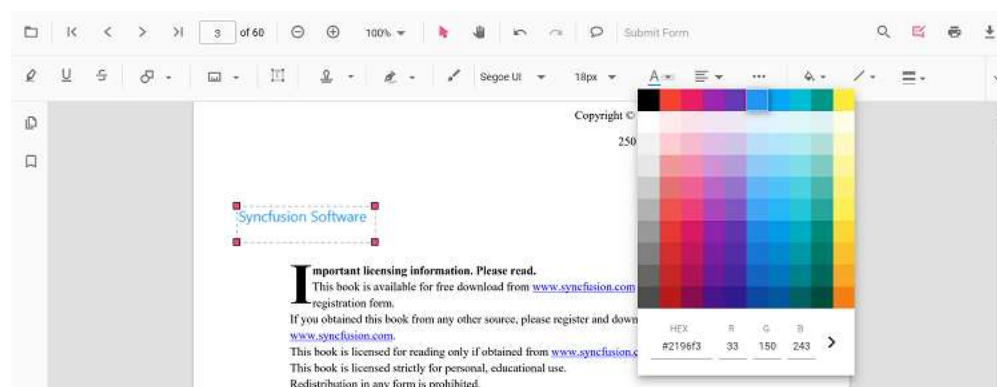
Editing font size

The font size of the annotation can be edited by selecting the desired size in the Font Size tool.



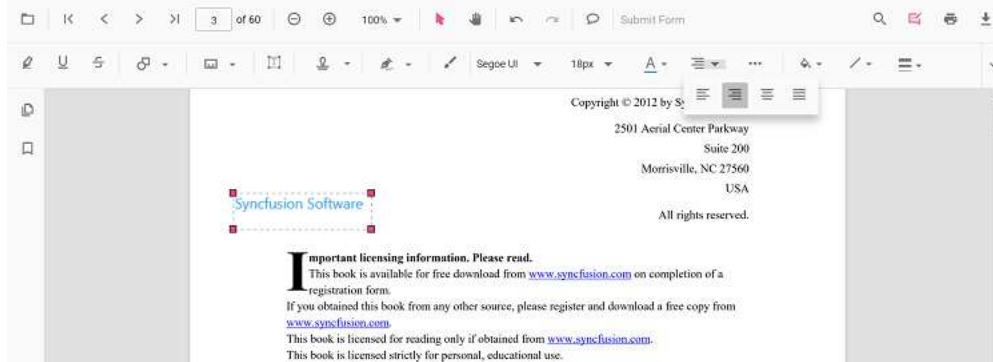
Editing font color

The font color of the annotation can be edited using the color palette provided in the Font Color tool.



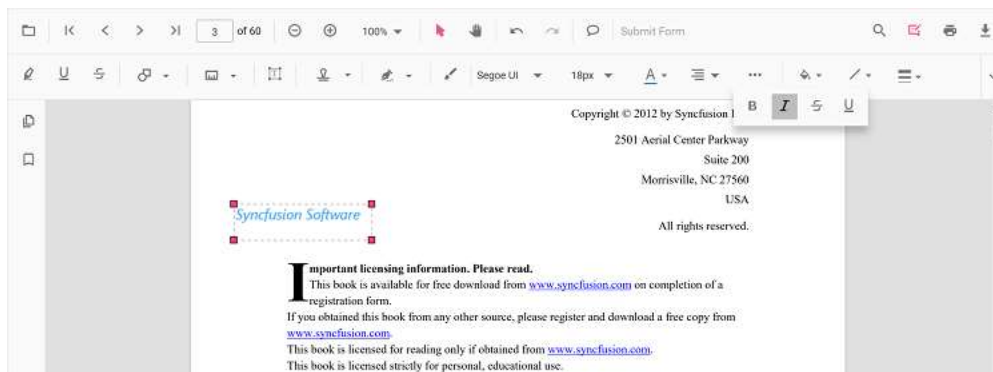
Editing the text alignment

The text in the annotation can be aligned by selecting the desired styles in the drop-down pop-up in the Text Align tool.



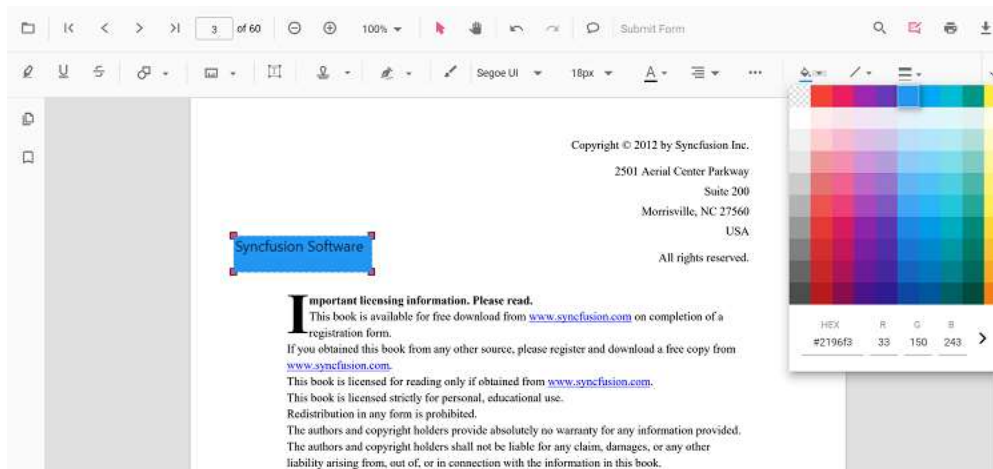
Editing text styles

The style of the text in the annotation can be edited by selecting the desired styles in the drop-down pop-up in the Font Style tool.



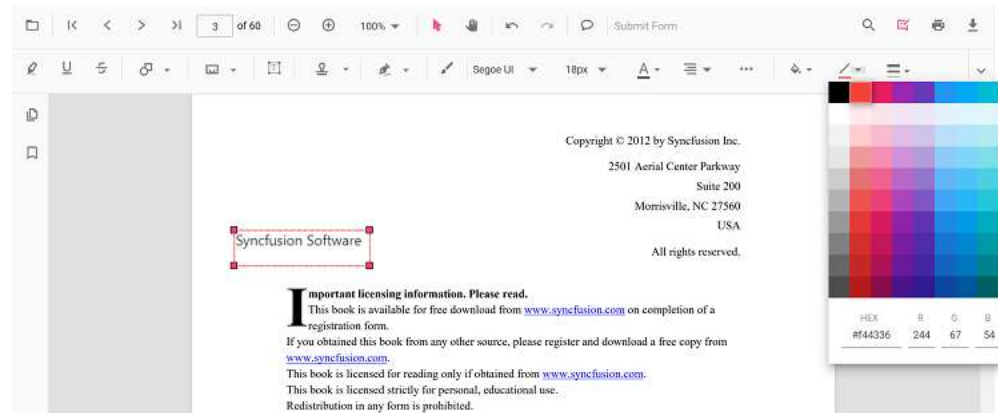
Editing fill color

The fill color of the annotation can be edited using the color palette provided in the Edit Color tool.



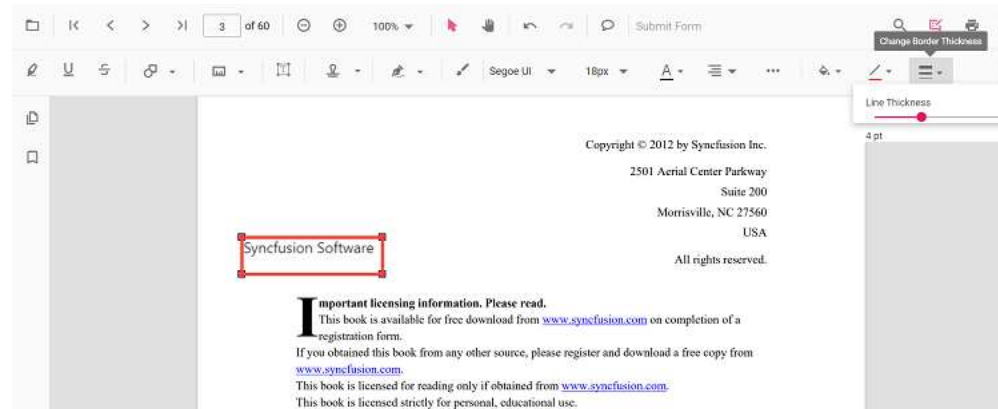
Editing stroke color

The stroke color of the annotation can be edited using the color palette provided in the Edit Stroke Color tool.



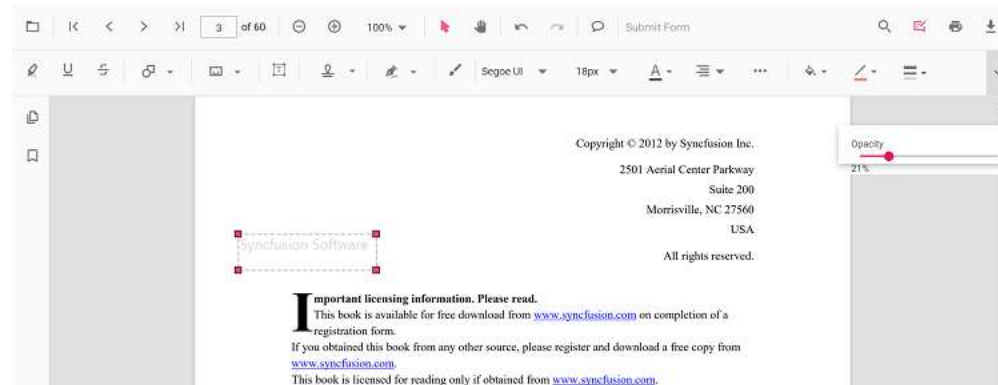
Editing thickness

The border thickness of the annotation can be edited using the range slider provided in the Edit Thickness tool.



Editing opacity

The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



Move the free text annotation programmatically

The PDF Viewer library allows you to move the free text annotation in the PDF Viewer control programmatically using the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method to move the free text annotation programmatically:

```
`html
```

```
<button (click)="moveFreeText()">Move the Free Text</button>
```

```
,
```

```
`typescript
```

```
moveFreeText() {
  var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  for (let i = 0; i < viewer.annotationCollection.length; i++)
  {
    if (viewer.annotationCollection[i].subject === "Text Box") {
      var width = viewer.annotationCollection[i].bounds.width;
      var height = viewer.annotationCollection[i].bounds.height;
      viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width, height: height };
      viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
    }
  }
}
```

```
,
```

Find the sample [how to move the free text annotation programmatically](#)

Get the newly added free text annotation ID

To get the ID of a newly added free text annotation in the Syncfusion PDF viewer, you can use the **annotationAdd()** event. This event is triggered whenever a new annotation is added to the PDF document, and it provides the `annotationAddEventHandler` object as a parameter. You can access the ID of the new annotation through the `AnnotationID` property of the `annotationAddEventHandler` object.

Here is an example of how you can use the **annotationAdd()** event to get the ID of a new free text annotation:

```
`typescript
```

```
public annotationAddEventHandler(args) {
  if (args.annotationType === 'FreeText') {
    console.log('annotationId:' + args.annotationId);
  }
}
```

```
,
```

Find the sample [how to get the newly added free text annotation id](#)

Change the content of an existing Free text annotation programmatically

To change the content of an existing free text annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method to change the content of a free text annotation:

```
`html
<button (click)="changeContent()">Change Content</button>
`

`typescript
changeContent() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++) {
if (viewer.annotationCollection[i].subject === 'Text Box') {
viewer.annotationCollection[i].dynamicText = 'syncfusion';
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
```

Find the sample [how to change the content of an existing free text annotation programmatically](#)

Setting default properties during control initialization

The properties of the free text annotation can be set before creating the control using the FreeTextSettings.

After editing the default values, they will be changed to the selected values.

Refer to the following code sample to set the default free text annotation settings.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// Specifies the template string for the PDF Viewer component.
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[freeTextSettings]='freeTextSettings'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
```

```
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  public freeTextSettings = { fillColor: 'green', borderColor: 'blue',
fontColor: 'yellow' };
  ngOnInit(): void {
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[freeTextSettings]='freeTextSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public freeTextSettings = { fillColor: 'green', borderColor: 'blue',
fontColor: 'yellow' };
  ngOnInit(): void {
  }
}
```

You can also enable the autofit support for free text annotation by using the `enableAutoFit` boolean property in `freeTextSettings` as below. The width of the free text rectangle box will be increased based on the text added to it.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import {
  PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
  MagnificationService, ThumbnailViewService, ToolbarService,
  NavigationService, TextSearchService, TextSelectionService,
  PrintService, AnnotationService, FormDesignerService, FormFieldsService,
  PageOrganizerService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specify the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
    <ejs-pdfviewer id="pdfViewer"
    [documentPath]='document'
    [freeTextSettings]='freeTextSettings'
    [resourceUrl]='resource'
    style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService,
    PageOrganizerService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
  succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
  pdfviewer-lib";
  public freeTextSettings = { enableAutoFit: true };
  ngOnInit(): void {
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
  MagnificationService, ThumbnailViewService, ToolbarService,
  NavigationService, TextSearchService, TextSelectionService,
  PrintService, AnnotationService, FormDesignerService, FormFieldsService,
  PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specify the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
    <ejs-pdfviewer id="pdfViewer"
    [serviceUrl]='service'
    [documentPath]='document'`
```

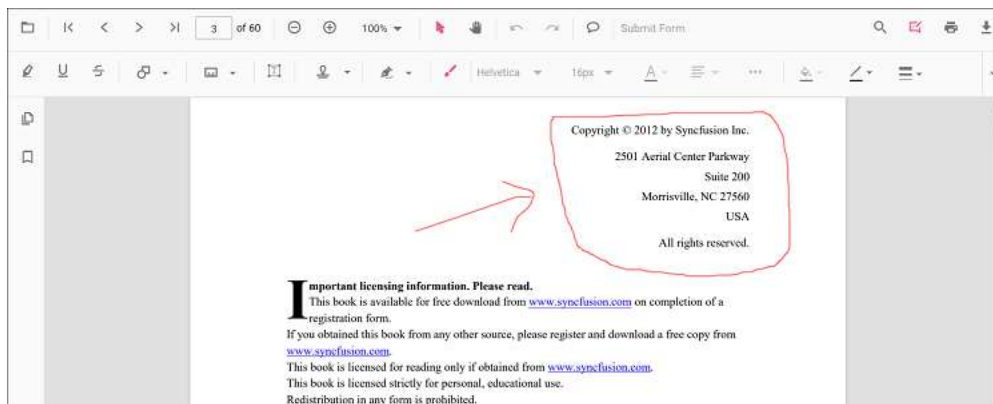


```
[freeTextSettings]='freeTextSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public freeTextSettings = { enableAutoFit: true };
ngOnInit(): void {
}
}
```

[View sample in GitHub](#)

Ink Annotation in Angular PDF Viewer component

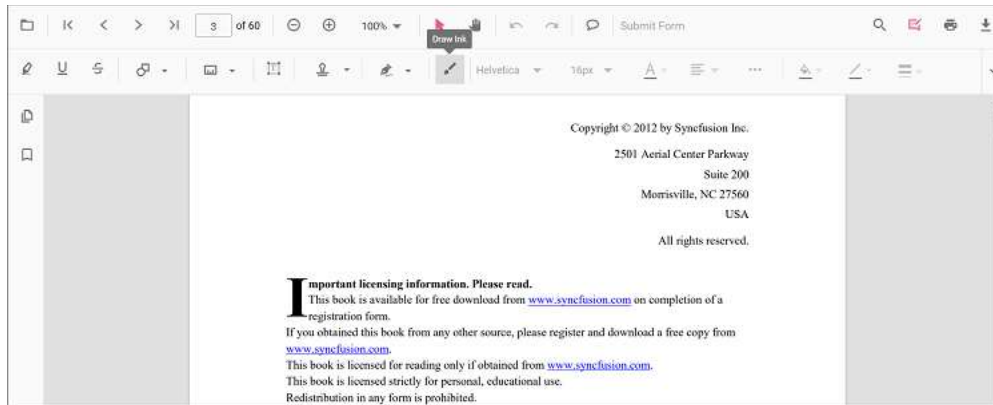
The PDF Viewer control provides the options to add, edit, and delete the ink annotations.



Adding an ink annotation to the PDF document

The ink annotations can be added to the PDF document using the annotation toolbar.

- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Draw Ink** button in the annotation toolbar. It enables the ink annotation mode.
- You can draw anything over the pages of the PDF document.



Refer to the following code sample to switch to the ink annotation mode.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService,
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// Specifies the template string for the PDF Viewer component.
template: `<button (click)="addAnnotation()">Draw Ink</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl] = 'resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService, FormDesignerService,
FormFieldsService, AnnotationService, PageOrganizerService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfviewer.annotationModule.setAnnotationMode("Ink");
}
}
```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService,
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<button (click)="addAnnotation()">Draw Ink</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService, FormDesignerService,
FormFieldsService, AnnotationService, PageOrganizerService]
})
export class AppComponent implements OnInit {
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  ngOnInit(): void {
  }
  addAnnotation() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.annotationModule.setAnnotationMode("Ink");
  }
}

```

Adding a Ink annotation to the PDF document Programmatically

With the PDF Viewer library, you can add a Ink annotation to the PDF Viewer control programmatically using the [addAnnotation\(\)](#) method.

Here's a example of how you can utilize the **addAnnotation()** method to include a Ink annotation programmatically

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, InkAnnotationSettings } from
 '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component

```

```

template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add Ink annotation
Programmatically</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Ink", {
offset: { x: 150, y: 100 },
pageNumber: 1,
width: 200,
height: 60,
path:
' [{"command\":"M", "x\":244.83334350585938, "y\":982.0000305175781}, {"com
mand\":"L", "x\":244.83334350585938, "y\":982.0000305175781}, {"command\":"
L", "x\":250.83334350585938, "y\":953.3333435058594}, {"command\":"L", "x
\":252.83334350585938, "y\":946.0000305175781}, {"command\":"L", "x\":254.1
6668701171875, "y\":940.6667175292969}, {"command\":"L", "x\":256.833343505
8594, "y\":931.3333435058594}, {"command\":"L", "x\":257.5, "y\":929.333343
5058594}, {"command\":"L", "x\":258.8333435058594, "y\":926.6667175292969},
{"command\":"L", "x\":259.5, "y\":924.0000305175781}, {"command\":"L", "
x\":259.5, "y\":922.6667175292969}, {"command\":"L", "x\":258.8333435058594
, "y\":922.0000305175781}, {"command\":"L", "x\":258.16668701171875, "y\":9
22.0000305175781}, {"command\":"L", "x\":256.8333435058594, "y\":922.000030
5175781}, {"command\":"L", "x\":256.16668701171875, "y\":922.6667175292969}
, {"command\":"L", "x\":254.83334350585938, "y\":923.3333435058594}, {"comm
and\":"L", "x\":254.16668701171875, "y\":923.3333435058594}, {"command\":"
L", "x\":253.5, "y\":923.3333435058594}, {"command\":"L", "x\":252.8333435
0585938, "y\":925.3333435058594}, {"command\":"L", "x\":252.83334350585938,
"y\":927.3333435058594}, {"command\":"L", "x\":252.83334350585938, "y\":93
6.0000305175781}, {"command\":"L", "x\":253.5, "y\":940.6667175292969}, {"c
ommand\":"L", "x\":254.83334350585938, "y\":944.6667175292969}, {"command\
":"L", "x\":260.16668701171875, "y\":952.0000305175781}, {"command\":"L", \
"x\":264.16668701171875, "y\":954.0000305175781}, {"command\":"L", "x\":274
.16668701171875, "y\":958.6667175292969}, {"command\":"L", "x\":278.1666870
1171875, "y\":960.0000305175781}, {"command\":"L", "x\":281.5, "y\":961.333
3435058594}, {"command\":"L", "x\":285.5, "y\":964.6667175292969}, {"comman
d\":"L", "x\":286.8333740234375, "y\":967.3333435058594}, {"command\":"L"

```

,\"x\":286.8333740234375,\"y\":970.0000305175781},{\"command\":\"L\", \"x\":282.8333740234375,\"y\":978.6667175292969},{\"command\":\"L\", \"x\":278.16668701171875,\"y\":983.3333435058594},{\"command\":\"L\", \"x\":266.16668701171875,\"y\":991.3333435058594},{\"command\":\"L\", \"x\":259.5,\"y\":993.3333435058594},{\"command\":\"L\", \"x\":252.16668701171875,\"y\":994.0000305175781},{\"command\":\"L\", \"x\":240.83334350585938,\"y\":991.3333435058594},{\"command\":\"L\", \"x\":236.16668701171875,\"y\":988.6667175292969},{\"command\":\"L\", \"x\":230.16668701171875,\"y\":982.6667175292969},{\"command\":\"L\", \"x\":228.83334350585938,\"y\":980.6667175292969},{\"command\":\"L\", \"x\":228.16668701171875,\"y\":978.6667175292969},{\"command\":\"L\", \"x\":228.83334350585938,\"y\":974.6667175292969},{\"command\":\"L\", \"x\":230.16668701171875,\"y\":973.3333435058594},{\"command\":\"L\", \"x\":236.16668701171875,\"y\":971.3333435058594},{\"command\":\"L\", \"x\":240.83334350585938,\"y\":971.3333435058594},{\"command\":\"L\", \"x\":246.16668701171875,\"y\":972.0000305175781},{\"command\":\"L\", \"x\":257.5,\"y\":974.6667175292969},{\"command\":\"L\", \"x\":262.8333435058594,\"y\":976.0000305175781},{\"command\":\"L\", \"x\":269.5,\"y\":977.3333435058594},{\"command\":\"L\", \"x\":276.16668701171875,\"y\":978.6667175292969},{\"command\":\"L\", \"x\":279.5,\"y\":978.0000305175781},{\"command\":\"L\", \"x\":285.5,\"y\":976.6667175292969},{\"command\":\"L\", \"x\":288.16668701171875,\"y\":974.6667175292969},{\"command\":\"L\", \"x\":292.8333740234375,\"y\":969.3333435058594},{\"command\":\"L\", \"x\":293.5,\"y\":966.6667175292969},{\"command\":\"L\", \"x\":294.16668701171875,\"y\":964.0000305175781},{\"command\":\"L\", \"x\":293.5,\"y\":960.0000305175781},{\"command\":\"L\", \"x\":293.5,\"y\":958.0000305175781},{\"command\":\"L\", \"x\":292.8333740234375,\"y\":956.6667175292969},{\"command\":\"L\", \"x\":291.5,\"y\":954.6667175292969},{\"command\":\"L\", \"x\":291.5,\"y\":954.0000305175781},{\"command\":\"L\", \"x\":291.5,\"y\":953.3333435058594},{\"command\":\"L\", \"x\":291.5,\"y\":954.0000305175781},{\"command\":\"L\", \"x\":292.16668701171875,\"y\":954.6667175292969},{\"command\":\"L\", \"x\":292.8333740234375,\"y\":956.0000305175781},{\"command\":\"L\", \"x\":294.16668701171875,\"y\":961.3333435058594},{\"command\":\"L\", \"x\":295.5,\"y\":964.6667175292969},{\"command\":\"L\", \"x\":297.5,\"y\":969.3333435058594},{\"command\":\"L\", \"x\":298.8333740234375,\"y\":970.6667175292969},{\"command\":\"L\", \"x\":301.5,\"y\":970.0000305175781},{\"command\":\"L\", \"x\":304.16668701171875,\"y\":968.6667175292969},{\"command\":\"L\", \"x\":305.5,\"y\":966.0000305175781},{\"command\":\"L\", \"x\":308.8333740234375,\"y\":960.0000305175781},{\"command\":\"L\", \"x\":310.16668701171875,\"y\":957.3333435058594},{\"command\":\"L\", \"x\":310.8333740234375,\"y\":956.0000305175781},{\"command\":\"L\", \"x\":310.8333740234375,\"y\":954.6667175292969},{\"command\":\"L\", \"x\":310.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\", \"x\":311.5,\"y\":956.0000305175781},{\"command\":\"L\", \"x\":312.8333740234375,\"y\":959.3333435058594},{\"command\":\"L\", \"x\":316.16668701171875,\"y\":968.0000305175781},{\"command\":\"L\", \"x\":317.5,\"y\":972.6667175292969},{\"command\":\"L\", \"x\":318.16668701171875,\"y\":977.3333435058594},{\"command\":\"L\", \"x\":319.5,\"y\":983.3333435058594},{\"command\":\"L\", \"x\":319.5,\"y\":986.0000305175781},{\"command\":\"L\", \"x\":319.5,\"y\":988.0000305175781},{\"command\":\"L\", \"x\":318.8333740234375,\"y\":988.6667175292969},{\"command\":\"L\", \"x\":316.16668701171875,\"y\":987.3333435058594},{\"command\":\"L\", \"x\":314.8333740234375,\"y\":985.3333435058594},{\"command\":\"L\", \"x\":314.16668701171875,\"y\":980.6667175292969},{\"command\":\"L\", \"x\":314.8333740234375,\"y\":974.6667175292969},{\"command\":\"L\", \"x\":316.16668701171875,\"y\":969.3333435058594},{\"command\":\"L\", \"x\":319.5,\"y\":960.6667175292969},{\"command\":\"L\", \"x\":320.16668701171875,\"y\":957.3333435058594},{\"command\":\"L\", \"x\":321.5,\"y\":955.3333435058594},{\"command\":\"L\", \"x\":322.16668701171875,\"y\":953.3333435058594},{\"command\":\"L\", \"x\":322.8333740234375,\"y\":952.6667175292969},{\"command\":\"L\", \"x\":324.16668701171875,\"y\":952.6667175292969},{\"command\":\"L\", \"x\":324.8

333740234375,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":326.8333740234375,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":328.16668701171875,\"y\":958.0000305175781},{\"command\":\"L\",\"x\":328.8333740234375,\"y\":960.0000305175781},{\"command\":\"L\",\"x\":329.5,\"y\":962.0000305175781},{\"command\":\"L\",\"x\":330.16668701171875,\"y\":962.0000305175781},{\"command\":\"L\",\"x\":330.16668701171875,\"y\":962.6667175292969},{\"command\":\"L\",\"x\":330.16668701171875,\"y\":962.0000305175781},{\"command\":\"L\",\"x\":330.8333740234375,\"y\":960.0000305175781},{\"command\":\"L\",\"x\":331.5,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":332.8333740234375,\"y\":952.0000305175781},{\"command\":\"L\",\"x\":333.5,\"y\":950.0000305175781},{\"command\":\"L\",\"x\":334.8333740234375,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":335.5,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":336.16668701171875,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":337.5,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":338.8333740234375,\"y\":952.0000305175781},{\"command\":\"L\",\"x\":340.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":341.5,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":342.8333740234375,\"y\":954.6667175292969},{\"command\":\"L\",\"x\":344.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":346.8333740234375,\"y\":952.6667175292969},{\"command\":\"L\",\"x\":349.5,\"y\":949.3333435058594},{\"command\":\"L\",\"x\":350.8333740234375,\"y\":948.0000305175781},{\"command\":\"L\",\"x\":351.5,\"y\":946.6667175292969},{\"command\":\"L\",\"x\":352.8333740234375,\"y\":944.0000305175781},{\"command\":\"L\",\"x\":354.16668701171875,\"y\":942.0000305175781},{\"command\":\"L\",\"x\":354.8333740234375,\"y\":942.0000305175781},{\"command\":\"L\",\"x\":354.8333740234375,\"y\":942.6667175292969},{\"command\":\"L\",\"x\":354.16668701171875,\"y\":943.3333435058594},{\"command\":\"L\",\"x\":354.16668701171875,\"y\":946.6667175292969},{\"command\":\"L\",\"x\":354.16668701171875,\"y\":950.0000305175781},{\"command\":\"L\",\"x\":355.5,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":356.16668701171875,\"y\":957.3333435058594},{\"command\":\"L\",\"x\":358.16668701171875,\"y\":959.3333435058594},{\"command\":\"L\",\"x\":360.16668701171875,\"y\":958.0000305175781},{\"command\":\"L\",\"x\":364.16668701171875,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":370.8333740234375,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":373.5,\"y\":943.3333435058594},{\"command\":\"L\",\"x\":375.5,\"y\":937.3333435058594},{\"command\":\"L\",\"x\":376.16668701171875,\"y\":933.3333435058594},{\"command\":\"L\",\"x\":376.8333740234375,\"y\":931.3333435058594},{\"command\":\"L\",\"x\":376.8333740234375,\"y\":930.0000305175781},{\"command\":\"L\",\"x\":376.8333740234375,\"y\":929.3333435058594},{\"command\":\"L\",\"x\":376.16668701171875,\"y\":930.0000305175781},{\"command\":\"L\",\"x\":375.5,\"y\":932.0000305175781},{\"command\":\"L\",\"x\":375.5,\"y\":937.3333435058594},{\"command\":\"L\",\"x\":374.8333740234375,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":374.8333740234375,\"y\":960.6667175292969},{\"command\":\"L\",\"x\":375.5,\"y\":966.0000305175781},{\"command\":\"L\",\"x\":377.5,\"y\":974.6667175292969},{\"command\":\"L\",\"x\":378.16668701171875,\"y\":977.3333435058594},{\"command\":\"L\",\"x\":380.8333740234375,\"y\":981.3333435058594},{\"command\":\"L\",\"x\":382.16668701171875,\"y\":982.6667175292969},{\"command\":\"L\",\"x\":383.5,\"y\":982.6667175292969},{\"command\":\"L\",\"x\":387.5,\"y\":982.6667175292969},{\"command\":\"L\",\"x\":389.5,\"y\":980.6667175292969},{\"command\":\"L\",\"x\":392.16668701171875,\"y\":976.6667175292969},{\"command\":\"L\",\"x\":392.8333740234375,\"y\":973.3333435058594},{\"command\":\"L\",\"x\":392.16668701171875,\"y\":970.0000305175781},{\"command\":\"L\",\"x\":388.8333740234375,\"y\":965.3333435058594},{\"command\":\"L\",\"x\":385.5,\"y\":964.0000305175781},{\"command\":\"L\",\"x\":382.8333740234375,\"y\":964.0000305175781},{\"command\":\"L\",\"x\":377.5,\"y\":964.0000305175781},{\"command\":\"L\",\"x\":375.5,\"y\":964.6667175292969},{\"command\":\"L\",\"x\":373.5,\"y\":965.3333435058594},{\"command\":\"L\",\"x\":374.8333740234375,\"y\":963.3333435058594},{\"command\":\"L\",\"x\":376.83337402343

75,\"y\":961.3333435058594},{\"command\":\"L\",\"x\":382.16668701171875,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":384.16668701171875,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":387.5,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":388.16668701171875,\"y\":952.0000305175781},{\"command\":\"L\",\"x\":388.16668701171875,\"y\":952.6667175292969},{\"command\":\"L\",\"x\":388.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":388.8333740234375,\"y\":954.6667175292969},{\"command\":\"L\",\"x\":389.5,\"y\":959.3333435058594},{\"command\":\"L\",\"x\":389.5,\"y\":960.6667175292969},{\"command\":\"L\",\"x\":390.16668701171875,\"y\":961.3333435058594},{\"command\":\"L\",\"x\":390.8333740234375,\"y\":960.6667175292969},{\"command\":\"L\",\"x\":393.5,\"y\":958.0000305175781},{\"command\":\"L\",\"x\":396.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":398.16668701171875,\"y\":952.0000305175781},{\"command\":\"L\",\"x\":400.16668701171875,\"y\":949.3333435058594},{\"command\":\"L\",\"x\":400.16668701171875,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":400.8333740234375,\"y\":948.0000305175781},{\"command\":\"L\",\"x\":400.8333740234375,\"y\":947.3333435058594},{\"command\":\"L\",\"x\":401.5,\"y\":948.0000305175781},{\"command\":\"L\",\"x\":402.16668701171875,\"y\":949.3333435058594},{\"command\":\"L\",\"x\":403.5,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":404.8333740234375,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":406.16668701171875,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":407.5,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":410.16668701171875,\"y\":952.0000305175781},{\"command\":\"L\",\"x\":412.16668701171875,\"y\":949.3333435058594},{\"command\":\"L\",\"x\":414.16668701171875,\"y\":944.6667175292969},{\"command\":\"L\",\"x\":414.16668701171875,\"y\":942.0000305175781},{\"command\":\"L\",\"x\":414.16668701171875,\"y\":940.6667175292969},{\"command\":\"L\",\"x\":414.16668701171875,\"y\":938.6667175292969},{\"command\":\"L\",\"x\":414.16668701171875,\"y\":938.0000305175781},{\"command\":\"L\",\"x\":415.5,\"y\":939.3333435058594},{\"command\":\"L\",\"x\":418.8333740234375,\"y\":942.6667175292969},{\"command\":\"L\",\"x\":420.16668701171875,\"y\":945.3333435058594},{\"command\":\"L\",\"x\":421.5,\"y\":946.6667175292969},{\"command\":\"L\",\"x\":422.8333740234375,\"y\":950.0000305175781},{\"command\":\"L\",\"x\":423.5,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":423.5,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":422.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":421.5,\"y\":955.3333435058594},{\"command\":\"L\",\"x\":421.5,\"y\":956.0000305175781},{\"command\":\"L\",\"x\":422.16668701171875,\"y\":954.6667175292969},{\"command\":\"L\",\"x\":422.8333740234375,\"y\":954.0000305175781},{\"command\":\"L\",\"x\":424.8333740234375,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":425.5,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":428.16668701171875,\"y\":945.3333435058594},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":943.3333435058594},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":942.6667175292969},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":943.3333435058594},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":945.3333435058594},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":948.0000305175781},{\"command\":\"L\",\"x\":428.8333740234375,\"y\":950.0000305175781},{\"command\":\"L\",\"x\":429.5,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":430.16668701171875,\"y\":953.3333435058594},{\"command\":\"L\",\"x\":432.8333740234375,\"y\":952.6667175292969},{\"command\":\"L\",\"x\":434.8333740234375,\"y\":950.6667175292969},{\"command\":\"L\",\"x\":437.5,\"y\":948.6667175292969},{\"command\":\"L\",\"x\":440.16668701171875,\"y\":944.6667175292969},{\"command\":\"L\",\"x\":441.5,\"y\":942.6667175292969},{\"command\":\"L\",\"x\":442.16668701171875,\"y\":942.0000305175781},{\"command\":\"L\",\"x\":442.8333740234375,\"y\":941.3333435058594},{\"command\":\"L\",\"x\":442.8333740234375,\"y\":942.0000305175781},{\"command\":\"L\",\"x\":442.8333740234375,\"y\":943.3333435058594},{\"command\":\"L\",\"x\":442.8333740234375,\"y\":944.6667175292969},{\"command\":\"L\",\"x\":442.8333740234375,\"y\":946.0000305175781},{\"command\":\"L\",\"x\":443.5,\"y\":949.3333435058594},{\"command\":\"L\",\"x\":444.16668701171875,\"y\":950.6667175292969}

```

9},{\"command\": \"L\", \"x\": 445.5, \"y\": 950.6667175292969}, {\"command\": \"L\",
, \"x\": 447.5, \"y\": 950.6667175292969}, {\"command\": \"L\", \"x\": 450.1666870117
1875, \"y\": 948.6667175292969}, {\"command\": \"L\", \"x\": 452.16668701171875, \"y
\": 945.3333435058594}, {\"command\": \"L\", \"x\": 453.5, \"y\": 942.6667175292969}
, {\"command\": \"L\", \"x\": 452.8333740234375, \"y\": 938.6667175292969}, {\"comm
nd\": \"L\", \"x\": 452.16668701171875, \"y\": 937.3333435058594}, {\"command\": \"L
\", \"x\": 450.8333740234375, \"y\": 936.6667175292969}, {\"command\": \"L\", \"x\":
448.8333740234375, \"y\": 936.0000305175781}, {\"command\": \"L\", \"x\": 447.5, \"y
\": 936.6667175292969}, {\"command\": \"L\", \"x\": 446.16668701171875, \"y\": 937.3
333435058594}, {\"command\": \"L\", \"x\": 445.5, \"y\": 938.6667175292969}, {\"comm
and\": \"L\", \"x\": 445.5, \"y\": 939.3333435058594}, {\"command\": \"L\", \"x\": 446
.16668701171875, \"y\": 939.3333435058594}, {\"command\": \"L\", \"x\": 446.8333740
234375, \"y\": 939.3333435058594}, {\"command\": \"L\", \"x\": 452.16668701171875, \
\"y\": 937.3333435058594}, {\"command\": \"L\", \"x\": 454.8333740234375, \"y\": 936.
6667175292969}, {\"command\": \"L\", \"x\": 456.8333740234375, \"y\": 936.000030517
5781}, {\"command\": \"L\", \"x\": 459.5, \"y\": 936.6667175292969}, {\"command\": \"
L\", \"x\": 460.8333740234375, \"y\": 937.3333435058594}, {\"command\": \"L\", \"x\"
: 461.5, \"y\": 938.6667175292969}, {\"command\": \"L\", \"x\": 462.16668701171875, \
\"y\": 942.0000305175781}, {\"command\": \"L\", \"x\": 462.16668701171875, \"y\": 942
.6667175292969}, {\"command\": \"L\", \"x\": 462.16668701171875, \"y\": 944.0000305
175781}, {\"command\": \"L\", \"x\": 462.16668701171875, \"y\": 943.3333435058594},
{\"command\": \"L\", \"x\": 462.16668701171875, \"y\": 942.6667175292969}, {\"comm
nd\": \"L\", \"x\": 462.16668701171875, \"y\": 941.3333435058594}, {\"command\": \"L
\", \"x\": 462.8333740234375, \"y\": 938.6667175292969}, {\"command\": \"L\", \"x\":
464.16668701171875, \"y\": 935.3333435058594}, {\"command\": \"L\", \"x\": 465.5, \"
y\": 933.3333435058594}, {\"command\": \"L\", \"x\": 466.16668701171875, \"y\": 932.
6667175292969}, {\"command\": \"L\", \"x\": 467.5, \"y\": 933.3333435058594}, {\"com
mand\": \"L\", \"x\": 469.5, \"y\": 935.3333435058594}, {\"command\": \"L\", \"x\": 47
0.16668701171875, \"y\": 938.6667175292969}, {\"command\": \"L\", \"x\": 472.833374
0234375, \"y\": 943.3333435058594}, {\"command\": \"L\", \"x\": 472.8333740234375, \
\"y\": 944.6667175292969}, {\"command\": \"L\", \"x\": 474.16668701171875, \"y\": 944
.6667175292969}, {\"command\": \"L\", \"x\": 475.5, \"y\": 944.0000305175781}, {\"co
mmand\": \"L\", \"x\": 478.16668701171875, \"y\": 941.3333435058594}, {\"command\":
\"L\", \"x\": 481.5, \"y\": 937.3333435058594}, {\"command\": \"L\", \"x\": 484.83337
40234375, \"y\": 934.0000305175781}, {\"command\": \"L\", \"x\": 488.8333740234375,
\"y\": 929.3333435058594}, {\"command\": \"L\", \"x\": 489.5, \"y\": 928.00003051757
81}]'
} as InkAnnotationSettings);
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService, InkAnnotationSettings } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="addAnnotation()">Add Ink annotation
Programmatically</button>

```



```

<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
addAnnotation() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.annotation.addAnnotation("Ink", {
offset: { x: 150, y: 100 },
pageNumber: 1,
width: 200,
height: 60,
path:
' [{"command": "M", "x": 244.83334350585938, "y": 982.0000305175781}, {"command": "L", "x": 244.83334350585938, "y": 982.0000305175781}, {"command": "L", "x": 250.83334350585938, "y": 953.3333435058594}, {"command": "L", "x": 252.83334350585938, "y": 946.0000305175781}, {"command": "L", "x": 254.16668701171875, "y": 940.6667175292969}, {"command": "L", "x": 256.8333435058594, "y": 931.3333435058594}, {"command": "L", "x": 257.5, "y": 929.3333435058594}, {"command": "L", "x": 258.8333435058594, "y": 926.6667175292969}, {"command": "L", "x": 259.5, "y": 924.0000305175781}, {"command": "L", "x": 259.5, "y": 922.6667175292969}, {"command": "L", "x": 258.8333435058594, "y": 922.0000305175781}, {"command": "L", "x": 258.16668701171875, "y": 922.0000305175781}, {"command": "L", "x": 256.8333435058594, "y": 922.0000305175781}, {"command": "L", "x": 256.16668701171875, "y": 922.6667175292969}, {"command": "L", "x": 254.83334350585938, "y": 923.3333435058594}, {"command": "L", "x": 254.16668701171875, "y": 923.3333435058594}, {"command": "L", "x": 253.5, "y": 923.3333435058594}, {"command": "L", "x": 252.83334350585938, "y": 925.3333435058594}, {"command": "L", "x": 252.83334350585938, "y": 927.3333435058594}, {"command": "L", "x": 252.83334350585938, "y": 936.0000305175781}, {"command": "L", "x": 253.5, "y": 940.6667175292969}, {"command": "L", "x": 254.83334350585938, "y": 944.6667175292969}, {"command": "L", "x": 260.16668701171875, "y": 952.0000305175781}, {"command": "L", "x": 264.16668701171875, "y": 954.0000305175781}, {"command": "L", "x": 274.16668701171875, "y": 958.6667175292969}, {"command": "L", "x": 278.16668701171875, "y": 960.0000305175781}, {"command": "L", "x": 281.5, "y": 961.3333435058594}, {"command": "L", "x": 285.5, "y": 964.6667175292969}, {"command": "L", "x": 286.8333740234375, "y": 967.3333435058594}, {"command": "L", "x": 286.8333740234375, "y": 970.0000305175781}, {"command": "L", "x": 282.8333740234375, "y": 978.6667175292969}, {"command": "L", "x": 278.16668701171875, "y": 983.3333435058594}, {"command": "L", "x": 266.16668701171875,

```

```
\\"y\\":991.3333435058594},{\\"command\\":\\"L\\",\\"x\\":259.5,\\"y\\":993.3333435058594},{\\"command\\":\\"L\\",\\"x\\":252.16668701171875,\\"y\\":994.0000305175781},{\\"command\\":\\"L\\",\\"x\\":240.83334350585938,\\"y\\":991.3333435058594},{\\"command\\":\\"L\\",\\"x\\":236.16668701171875,\\"y\\":988.6667175292969},{\\"command\\":\\"L\\",\\"x\\":230.16668701171875,\\"y\\":982.6667175292969},{\\"command\\":\\"L\\",\\"x\\":228.83334350585938,\\"y\\":980.6667175292969},{\\"command\\":\\"L\\",\\"x\\":228.16668701171875,\\"y\\":978.6667175292969},{\\"command\\":\\"L\\",\\"x\\":228.83334350585938,\\"y\\":974.6667175292969},{\\"command\\":\\"L\\",\\"x\\":230.16668701171875,\\"y\\":973.3333435058594},{\\"command\\":\\"L\\",\\"x\\":236.16668701171875,\\"y\\":971.3333435058594},{\\"command\\":\\"L\\",\\"x\\":240.83334350585938,\\"y\\":971.3333435058594},{\\"command\\":\\"L\\",\\"x\\":246.16668701171875,\\"y\\":972.0000305175781},{\\"command\\":\\"L\\",\\"x\\":257.5,\\"y\\":974.6667175292969},{\\"command\\":\\"L\\",\\"x\\":262.8333435058594,\\"y\\":976.0000305175781},{\\"command\\":\\"L\\",\\"x\\":269.5,\\"y\\":977.3333435058594},{\\"command\\":\\"L\\",\\"x\\":276.16668701171875,\\"y\\":978.6667175292969},{\\"command\\":\\"L\\",\\"x\\":279.5,\\"y\\":978.0000305175781},{\\"command\\":\\"L\\",\\"x\\":285.5,\\"y\\":976.6667175292969},{\\"command\\":\\"L\\",\\"x\\":288.16668701171875,\\"y\\":974.6667175292969},{\\"command\\":\\"L\\",\\"x\\":292.8333740234375,\\"y\\":969.3333435058594},{\\"command\\":\\"L\\",\\"x\\":293.5,\\"y\\":966.6667175292969},{\\"command\\":\\"L\\",\\"x\\":294.16668701171875,\\"y\\":964.0000305175781},{\\"command\\":\\"L\\",\\"x\\":293.5,\\"y\\":960.0000305175781},{\\"command\\":\\"L\\",\\"x\\":293.5,\\"y\\":958.0000305175781},{\\"command\\":\\"L\\",\\"x\\":292.8333740234375,\\"y\\":956.6667175292969},{\\"command\\":\\"L\\",\\"x\\":291.5,\\"y\\":954.6667175292969},{\\"command\\":\\"L\\",\\"x\\":291.5,\\"y\\":954.0000305175781},{\\"command\\":\\"L\\",\\"x\\":291.5,\\"y\\":953.3333435058594},{\\"command\\":\\"L\\",\\"x\\":291.5,\\"y\\":954.0000305175781},{\\"command\\":\\"L\\",\\"x\\":292.16668701171875,\\"y\\":954.6667175292969},{\\"command\\":\\"L\\",\\"x\\":292.8333740234375,\\"y\\":956.0000305175781},{\\"command\\":\\"L\\",\\"x\\":294.16668701171875,\\"y\\":961.3333435058594},{\\"command\\":\\"L\\",\\"x\\":295.5,\\"y\\":964.6667175292969},{\\"command\\":\\"L\\",\\"x\\":297.5,\\"y\\":969.3333435058594},{\\"command\\":\\"L\\",\\"x\\":298.8333740234375,\\"y\\":970.6667175292969},{\\"command\\":\\"L\\",\\"x\\":301.5,\\"y\\":970.0000305175781},{\\"command\\":\\"L\\",\\"x\\":304.16668701171875,\\"y\\":968.6667175292969},{\\"command\\":\\"L\\",\\"x\\":305.5,\\"y\\":966.0000305175781},{\\"command\\":\\"L\\",\\"x\\":308.8333740234375,\\"y\\":960.0000305175781},{\\"command\\":\\"L\\",\\"x\\":310.16668701171875,\\"y\\":957.3333435058594},{\\"command\\":\\"L\\",\\"x\\":310.8333740234375,\\"y\\":956.0000305175781},{\\"command\\":\\"L\\",\\"x\\":310.8333740234375,\\"y\\":954.6667175292969},{\\"command\\":\\"L\\",\\"x\\":310.8333740234375,\\"y\\":954.0000305175781},{\\"command\\":\\"L\\",\\"x\\":311.5,\\"y\\":956.0000305175781},{\\"command\\":\\"L\\",\\"x\\":312.8333740234375,\\"y\\":959.3333435058594},{\\"command\\":\\"L\\",\\"x\\":316.16668701171875,\\"y\\":968.0000305175781},{\\"command\\":\\"L\\",\\"x\\":317.5,\\"y\\":972.6667175292969},{\\"command\\":\\"L\\",\\"x\\":318.16668701171875,\\"y\\":977.3333435058594},{\\"command\\":\\"L\\",\\"x\\":319.5,\\"y\\":983.3333435058594},{\\"command\\":\\"L\\",\\"x\\":319.5,\\"y\\":986.0000305175781},{\\"command\\":\\"L\\",\\"x\\":319.5,\\"y\\":988.0000305175781},{\\"command\\":\\"L\\",\\"x\\":318.8333740234375,\\"y\\":988.0000305175781},{\\"command\\":\\"L\\",\\"x\\":318.16668701171875,\\"y\\":987.3333435058594},{\\"command\\":\\"L\\",\\"x\\":314.8333740234375,\\"y\\":985.3333435058594},{\\"command\\":\\"L\\",\\"x\\":314.16668701171875,\\"y\\":980.6667175292969},{\\"command\\":\\"L\\",\\"x\\":314.8333740234375,\\"y\\":974.6667175292969},{\\"command\\":\\"L\\",\\"x\\":316.16668701171875,\\"y\\":969.3333435058594},{\\"command\\":\\"L\\",\\"x\\":319.5,\\"y\\":960.6667175292969},{\\"command\\":\\"L\\",\\"x\\":320.16668701171875,\\"y\\":957.3333435058594},{\\"command\\":\\"L\\",\\"x\\":321.5,\\"y\\":955.3333435058594},{\\"command\\":\\"L\\",\\"x\\":322.16668701171875,\\"y\\":953.3333435058594},{\\"command\\":\\"L\\",\\"x\\":322.8333740234375,\\"y\\":952.6667175292969},{\\"command\\":\\"L\\",\\"x\\":324.16668701171875,\\"y\\":952.6667175292969},{\\"command\\":\\"L\\",\\"x\\":324.8333740234375,\\"y\\":953.3333435058594},{\\"command\\":\\"L\\",\\"x\\":326.8333740234375,\\"y\\":956.0000305175781},{\\"command\\":\\"L\\",\\"x\\":328.16668701171875,\\"y\\":958.0000305175781},{\\"command\\":\\"L\\",\\"x\\":328.8333740234375,\\"y\\":960.000
```

```
0305175781},{\"command\":\"L\", \"x\":329.5, \"y\":962.0000305175781},{\"command\":\"L\", \"x\":330.16668701171875, \"y\":962.0000305175781},{\"command\":\"L\", \"x\":330.16668701171875, \"y\":962.6667175292969},{\"command\":\"L\", \"x\":330.16668701171875, \"y\":962.0000305175781},{\"command\":\"L\", \"x\":330.8333740234375, \"y\":960.0000305175781},{\"command\":\"L\", \"x\":331.5, \"y\":956.0000305175781},{\"command\":\"L\", \"x\":332.8333740234375, \"y\":952.0000305175781},{\"command\":\"L\", \"x\":333.5, \"y\":950.0000305175781},{\"command\":\"L\", \"x\":334.8333740234375, \"y\":948.6667175292969},{\"command\":\"L\", \"x\":335.5, \"y\":948.6667175292969},{\"command\":\"L\", \"x\":336.16668701171875, \"y\":948.6667175292969},{\"command\":\"L\", \"x\":337.5, \"y\":950.6667175292969},{\"command\":\"L\", \"x\":338.8333740234375, \"y\":952.0000305175781},{\"command\":\"L\", \"x\":340.8333740234375, \"y\":954.0000305175781},{\"command\":\"L\", \"x\":341.5, \"y\":954.0000305175781},{\"command\":\"L\", \"x\":342.8333740234375, \"y\":954.6667175292969},{\"command\":\"L\", \"x\":344.8333740234375, \"y\":954.0000305175781},{\"command\":\"L\", \"x\":346.8333740234375, \"y\":952.6667175292969},{\"command\":\"L\", \"x\":349.5, \"y\":949.3333435058594},{\"command\":\"L\", \"x\":350.8333740234375, \"y\":948.0000305175781},{\"command\":\"L\", \"x\":351.5, \"y\":946.6667175292969},{\"command\":\"L\", \"x\":352.8333740234375, \"y\":944.0000305175781},{\"command\":\"L\", \"x\":352.8333740234375, \"y\":943.3333435058594},{\"command\":\"L\", \"x\":354.16668701171875, \"y\":942.0000305175781},{\"command\":\"L\", \"x\":354.8333740234375, \"y\":942.0000305175781},{\"command\":\"L\", \"x\":354.8333740234375, \"y\":942.6667175292969},{\"command\":\"L\", \"x\":354.16668701171875, \"y\":943.3333435058594},{\"command\":\"L\", \"x\":354.16668701171875, \"y\":946.6667175292969},{\"command\":\"L\", \"x\":354.16668701171875, \"y\":950.0000305175781},{\"command\":\"L\", \"x\":355.5, \"y\":956.0000305175781},{\"command\":\"L\", \"x\":356.16668701171875, \"y\":957.3333435058594},{\"command\":\"L\", \"x\":358.16668701171875, \"y\":959.3333435058594},{\"command\":\"L\", \"x\":360.16668701171875, \"y\":958.0000305175781},{\"command\":\"L\", \"x\":364.16668701171875, \"y\":956.0000305175781},{\"command\":\"L\", \"x\":370.8333740234375, \"y\":948.6667175292969},{\"command\":\"L\", \"x\":373.5, \"y\":943.3333435058594},{\"command\":\"L\", \"x\":375.5, \"y\":937.3333435058594},{\"command\":\"L\", \"x\":376.16668701171875, \"y\":933.3333435058594},{\"command\":\"L\", \"x\":376.8333740234375, \"y\":931.3333435058594},{\"command\":\"L\", \"x\":376.8333740234375, \"y\":930.0000305175781},{\"command\":\"L\", \"x\":376.8333740234375, \"y\":929.3333435058594},{\"command\":\"L\", \"x\":376.16668701171875, \"y\":930.0000305175781},{\"command\":\"L\", \"x\":375.5, \"y\":932.0000305175781},{\"command\":\"L\", \"x\":375.5, \"y\":937.3333435058594},{\"command\":\"L\", \"x\":374.8333740234375, \"y\":953.3333435058594},{\"command\":\"L\", \"x\":374.8333740234375, \"y\":960.6667175292969},{\"command\":\"L\", \"x\":375.5, \"y\":966.0000305175781},{\"command\":\"L\", \"x\":377.5, \"y\":974.6667175292969},{\"command\":\"L\", \"x\":378.16668701171875, \"y\":977.3333435058594},{\"command\":\"L\", \"x\":380.8333740234375, \"y\":981.3333435058594},{\"command\":\"L\", \"x\":382.16668701171875, \"y\":982.6667175292969},{\"command\":\"L\", \"x\":383.5, \"y\":982.6667175292969},{\"command\":\"L\", \"x\":387.5, \"y\":982.6667175292969},{\"command\":\"L\", \"x\":389.5, \"y\":980.6667175292969},{\"command\":\"L\", \"x\":392.16668701171875, \"y\":976.6667175292969},{\"command\":\"L\", \"x\":392.8333740234375, \"y\":973.3333435058594},{\"command\":\"L\", \"x\":392.16668701171875, \"y\":970.0000305175781},{\"command\":\"L\", \"x\":388.8333740234375, \"y\":965.3333435058594},{\"command\":\"L\", \"x\":385.5, \"y\":964.0000305175781},{\"command\":\"L\", \"x\":382.8333740234375, \"y\":964.0000305175781},{\"command\":\"L\", \"x\":377.5, \"y\":964.0000305175781},{\"command\":\"L\", \"x\":375.5, \"y\":964.6667175292969},{\"command\":\"L\", \"x\":373.5, \"y\":965.3333435058594},{\"command\":\"L\", \"x\":374.8333740234375, \"y\":963.3333435058594},{\"command\":\"L\", \"x\":376.8333740234375, \"y\":961.3333435058594},{\"command\":\"L\", \"x\":382.16668701171875, \"y\":956.0000305175781},{\"command\":\"L\", \"x\":384.16668701171875, \"y\":953.3333435058594},{\"command\":\"L\", \"x\":387.5, \"y\":950.6667175292969},{\"command\"
```

[illegible]

```

\"":945.3333435058594},{\"command\":\"L\", \"x\":453.5, \"y\":942.6667175292969}
, {\"command\":\"L\", \"x\":452.8333740234375, \"y\":938.6667175292969}, {\"comm
and\":\"L\", \"x\":452.16668701171875, \"y\":937.3333435058594}, {\"command\":\"L
\", \"x\":450.8333740234375, \"y\":936.6667175292969}, {\"command\":\"L\", \"x\":
448.8333740234375, \"y\":936.0000305175781}, {\"command\":\"L\", \"x\":447.5, \"y
\":936.6667175292969}, {\"command\":\"L\", \"x\":446.16668701171875, \"y\":937.3
333435058594}, {\"command\":\"L\", \"x\":445.5, \"y\":938.6667175292969}, {\"comm
and\":\"L\", \"x\":445.5, \"y\":939.3333435058594}, {\"command\":\"L\", \"x\":446
.16668701171875, \"y\":939.3333435058594}, {\"command\":\"L\", \"x\":446.8333740
234375, \"y\":939.3333435058594}, {\"command\":\"L\", \"x\":452.16668701171875, \
'y\":937.3333435058594}, {\"command\":\"L\", \"x\":454.8333740234375, \"y\":936.
6667175292969}, {\"command\":\"L\", \"x\":456.8333740234375, \"y\":936.000030517
5781}, {\"command\":\"L\", \"x\":459.5, \"y\":936.6667175292969}, {\"command\":\"L
\", \"x\":460.8333740234375, \"y\":937.3333435058594}, {\"command\":\"L\", \"x\"
:461.5, \"y\":938.6667175292969}, {\"command\":\"L\", \"x\":462.16668701171875, \
'y\":942.0000305175781}, {\"command\":\"L\", \"x\":462.16668701171875, \"y\":942
.6667175292969}, {\"command\":\"L\", \"x\":462.16668701171875, \"y\":944.0000305
175781}, {\"command\":\"L\", \"x\":462.16668701171875, \"y\":943.3333435058594},
{\"command\":\"L\", \"x\":462.16668701171875, \"y\":942.6667175292969}, {\"comm
and\":\"L\", \"x\":462.16668701171875, \"y\":941.3333435058594}, {\"command\":\"L
\", \"x\":462.8333740234375, \"y\":938.6667175292969}, {\"command\":\"L\", \"x\":
464.16668701171875, \"y\":935.3333435058594}, {\"command\":\"L\", \"x\":465.5, \
'y\":933.3333435058594}, {\"command\":\"L\", \"x\":466.16668701171875, \"y\":932.
6667175292969}, {\"command\":\"L\", \"x\":467.5, \"y\":933.3333435058594}, {\"com
mand\":\"L\", \"x\":469.5, \"y\":935.3333435058594}, {\"command\":\"L\", \"x\":47
0.16668701171875, \"y\":938.6667175292969}, {\"command\":\"L\", \"x\":472.833374
0234375, \"y\":943.3333435058594}, {\"command\":\"L\", \"x\":472.8333740234375, \
'y\":944.6667175292969}, {\"command\":\"L\", \"x\":474.16668701171875, \"y\":944
.6667175292969}, {\"command\":\"L\", \"x\":475.5, \"y\":944.0000305175781}, {\"co
mmand\":\"L\", \"x\":478.16668701171875, \"y\":941.3333435058594}, {\"command\":\"
L\", \"x\":481.5, \"y\":937.3333435058594}, {\"command\":\"L\", \"x\":484.83337
40234375, \"y\":934.0000305175781}, {\"command\":\"L\", \"x\":488.8333740234375,
\"y\":929.3333435058594}, {\"command\":\"L\", \"x\":489.5, \"y\":928.00003051757
81}}'
} as InkAnnotationSettings);
}
}

```

[Edit the existing Ink annotation programmatically](#)

To modify existing Ink annotation in the Syncfusion PDF viewer programmatically, you can use the **editAnnotation()** method.

Here is an example of how you can use the **editAnnotation()** method:

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component

```

```

template: `<div class="content-wrapper">
<button (click)="editAnnotation()">Edit ink annotation</button>
<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
editAnnotation() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++)
{
if (viewer.annotationCollection[i].shapeAnnotationType === "Ink") {
var width = viewer.annotationCollection[i].bounds.width;
var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotationCollection[i].strokeColor = "#0000FF";
viewer.annotationCollection[i].thickness = 2 ;
viewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService,
AnnotationService, PageOrganizerService } from '@syncfusion/ej2-angular-
pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<button (click)="editAnnotation()">Edit ink annotation</button>

```

```

<ejs-pdfviewer
id="pdfViewer"
[documentPath]='document'
[serviceUrl]='service'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
ngOnInit(): void {
}
editAnnotation() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (let i = 0; i < viewer.annotationCollection.length; i++)
{
if (viewer.annotationCollection[i].shapeAnnotationType === "Ink") {
var width = viewer.annotationCollection[i].bounds.width;
var height = viewer.annotationCollection[i].bounds.height;
viewer.annotationCollection[i].bounds = {x : 100, y: 100, width: width,
height: height };
viewer.annotationCollection[i].strokeColor = "#0000FF";
viewer.annotationCollection[i].thickness = 2 ;
viewer.annotationCollection[i].annotationSelectorSettings.resizerShape =
"Circle"
viewer.annotation.editAnnotation(viewer.annotationCollection[i]);
}
}
}
}
}

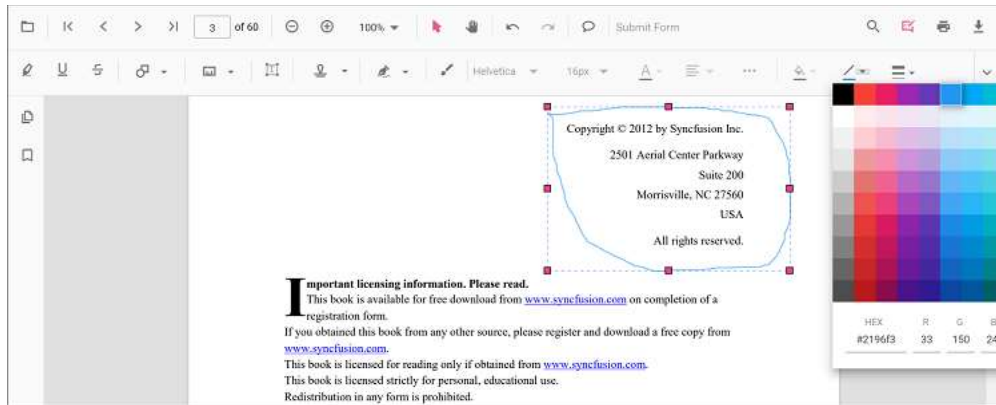
```

Editing the properties of the ink annotation

The stroke color, thickness, and opacity of the ink annotation can be edited using the Edit stroke color tool, Edit thickness tool, and Edit opacity tool in the annotation toolbar.

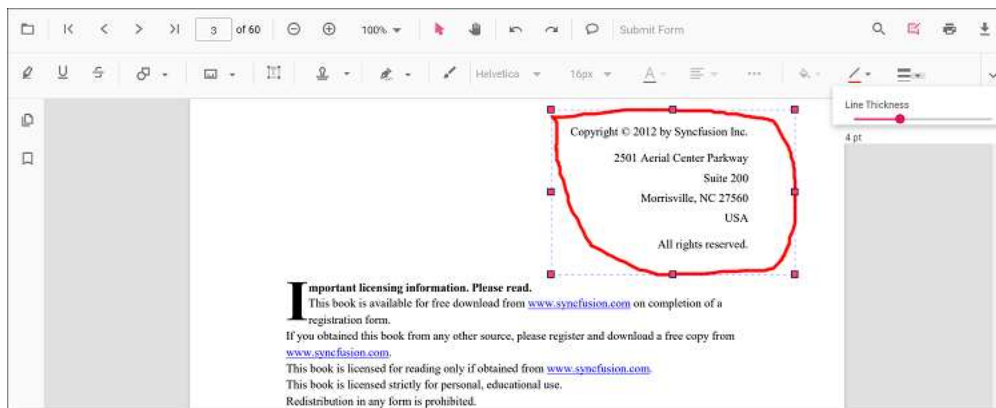
Editing stroke color

The stroke color of the annotation can be edited using the color palette provided in the Edit Stroke Color tool.



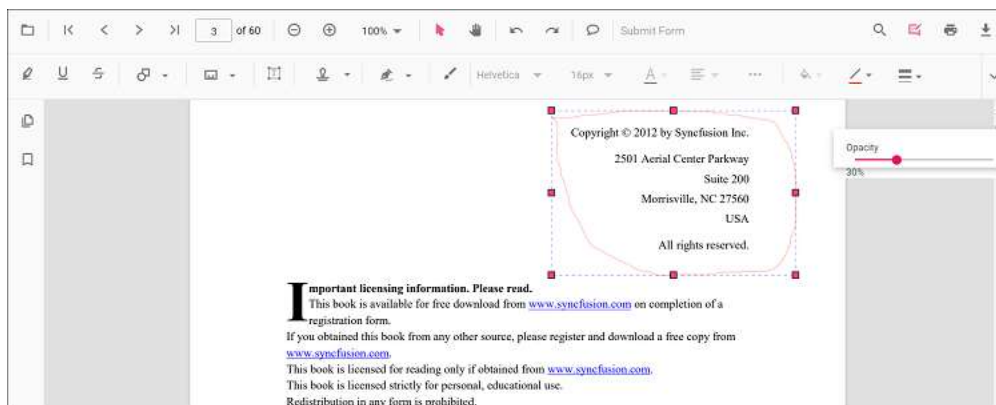
Editing thickness

The thickness of the border of the annotation can be edited using the range slider provided in the Edit Thickness tool.



Editing opacity

The opacity of the annotation can be edited using the range slider provided in the Edit Opacity tool.



Setting default properties during the control initialization

The properties of the ink annotation can be set before creating the control using the InkAnnotationSettings.

After editing the default values, they will be changed to the selected values.

Refer to the following code sample to set the default ink annotation settings.

STANDALONE


```

import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl] = 'resource'
[inkAnnotationSettings]='inkAnnotationSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib";
  public inkAnnotationSettings = { author: 'Syncfusion', strokeColor: 'green',
thickness: 3, opacity: 0.6 }
  ngOnInit(): void {
  }
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[inkAnnotationSettings]='inkAnnotationSettings'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,

```

```

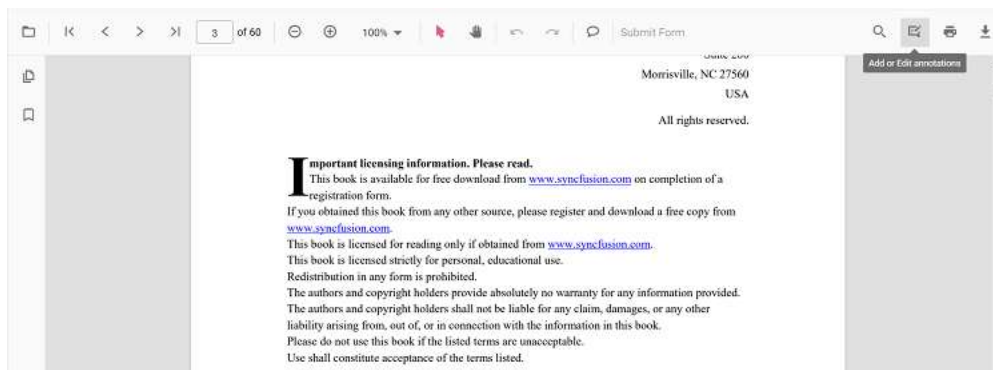
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  public service: string =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
  public inkAnnotationSettings = { author: 'Syncfusion', strokeColor: 'green',
    thickness: 3, opacity: 0.6 }
  ngOnInit(): void {
  }
}

```

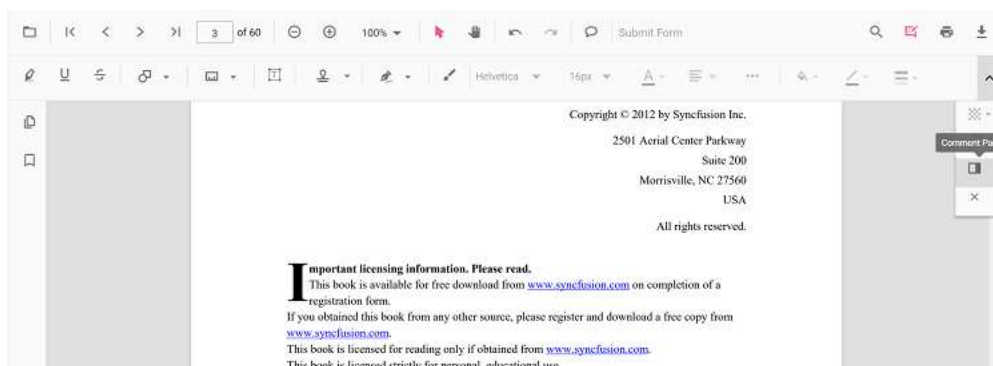
Import and export annotation

The PDF Viewer control provides the support to import and export annotations using JSON object in the PDF document.

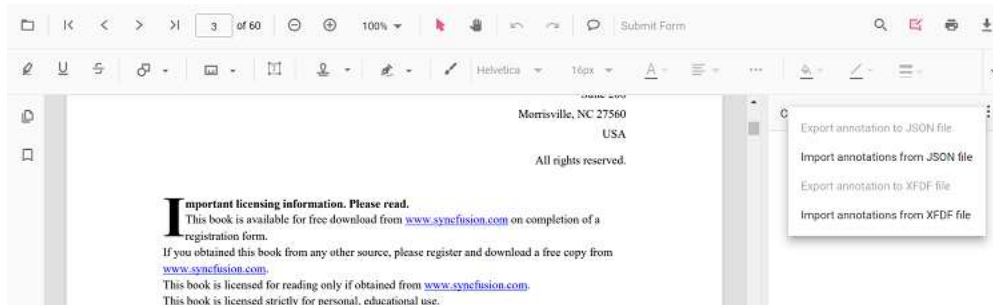
- Click the Add or Edit annotation button in the PDF Viewer toolbar.



- The annotation toolbar will appear.
- Click the Comment Panel button in the annotation toolbar.

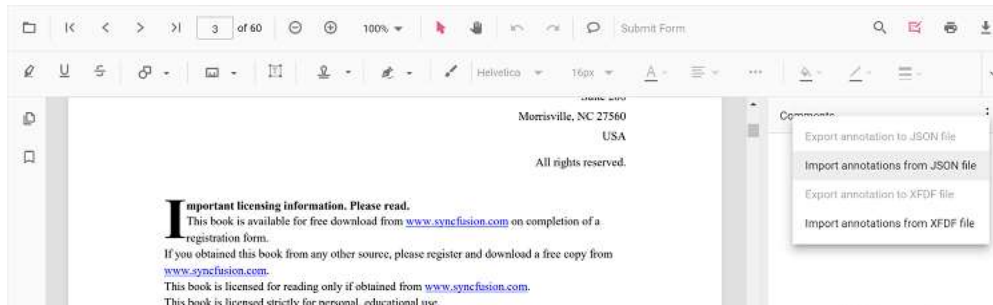


- The comments panel will displayed.
- Click the **More Option** button in the comment panel container.

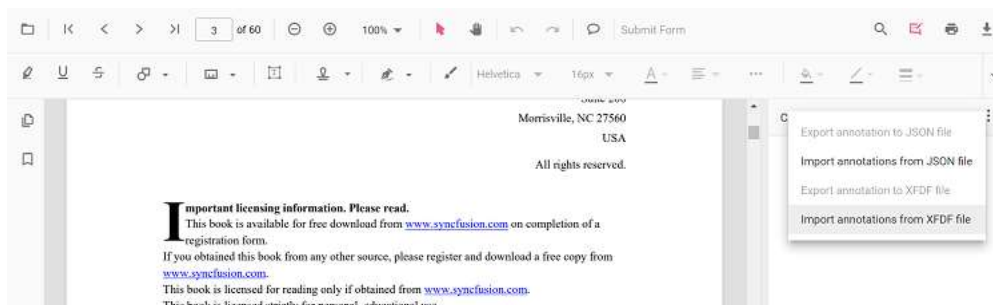


Importing annotation to the PDF document

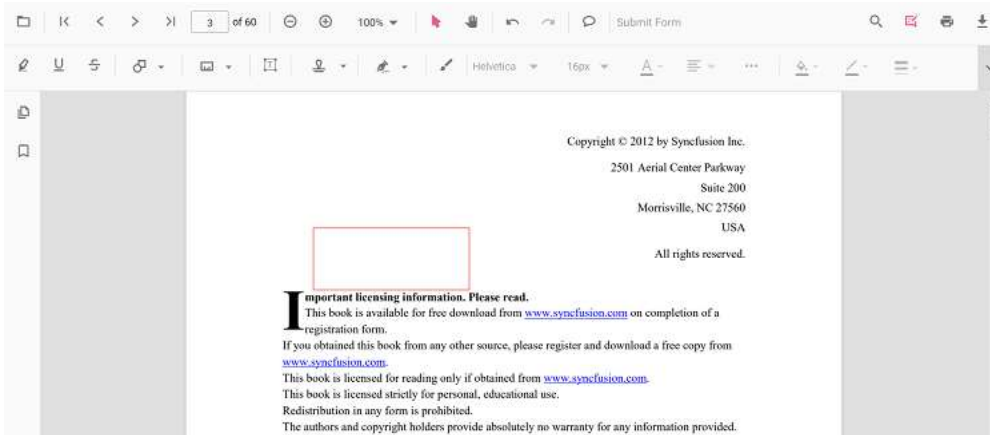
- Click the Add or Edit annotation button in the PDF Viewer toolbar.
- The annotation toolbar will appear.
- Click the Comment Panel button in the annotation toolbar.
- The comments panel will displayed.
- Click the **More Option** button in the comment panel container.
- Select the Import annotations from JSON file option to import annotations from a JSON file.



- Select the Import annotations from XDF file option to import annotations from an XDF file.



- Then the file explorer dialog will be opened. Choose the JSON file or the XDF file to be imported into the loaded PDF document.



Importing annotation using PDF Viewer API

You can import annotations using JSON object or JSON file in code behind like the below code snippet.

STANDALONE

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService],
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  //Local file path
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  OnImportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.importAnnotation({
      pdfAnnotation: {
        '0': {
          shapeAnnotation: [
            {
```

```

ShapeAnnotationType: 'Square',
Author: 'Guest',
AnnotationSelectorSettings: {
  selectionBorderColor: '',
  resizerBorderColor: 'black',
  resizerFillColor: '#FF4081',
  resizerSize: 8,
  selectionBorderThickness: 1,
  resizerShape: 'Square',
  selectorLineDashArray: [],
  resizerLocation: 3,
  resizerCursorType: null,
},
ModifiedDate: '5/19/2021, 11:44:51 AM',
Subject: 'Rectangle',
Note: '',
IsCommentLock: false,
StrokeColor: 'rgba(255,0,0,1)',
FillColor: 'rgba(255,255,255,0)',
Opacity: 1,
Bounds: {
  X: 154,
  Y: 71,
  Width: 299,
  Height: 173,
  Location: { X: 154, Y: 71 },
  Size: { IsEmpty: false, Width: 299, Height: 173 },
  Left: 154,
  Top: 71,
  Right: 453,
  Bottom: 244,
},
Thickness: 1,
BorderStyle: 'Solid',
BorderDashArray: 0,
RotateAngle: 'RotateAngle0',
IsCloudShape: false,
CloudIntensity: 0,
RectangleDifference: null,
VertexPoints: null,
LineHeadStart: null,
LineHeadEnd: null,
IsLocked: false,
AnnotName: 'e9ba2384-a202-45ff-0146-444fa0424dc5',
Comments: null,
State: '',
StateModel: '',
AnnotType: 'shape',
EnableShapeLabel: false,
LabelContent: null,
LabelFillColor: null,
LabelBorderColor: null,
FontColor: null,
FontSize: 0,
CustomData: null,
LabelBounds: {
  X: 0,

```

```

Y: 0,
Width: 0,
Height: 0,
Location: { X: 0, Y: 0 },
Size: { IsEmpty: true, Width: 0, Height: 0 },
Left: 0,
Top: 0,
Right: 0,
Bottom: 0,
},
LabelSettings: null,
AnnotationSettings: {
  minWidth: 0,
  maxWidth: 0,
  minHeight: 0,
  maxHeight: 0,
  isLock: false,
  isPrint: true,
},
AllowedInteractions: ['None'],
IsPrint: true,
ExistingCustomData: null,
},
{
  ShapeAnnotationType: 'Circle',
  Author: 'Guest',
  AnnotationSelectorSettings: {
    selectionBorderColor: '',
    resizerBorderColor: 'black',
    resizerFillColor: '#FF4081',
    resizerSize: 8,
    selectionBorderThickness: 1,
    resizerShape: 'Square',
    selectorLineDashArray: [],
    resizerLocation: 3,
    resizerCursorType: null,
  },
  ModifiedDate: '5/19/2021, 11:44:53 AM',
  Subject: 'Circle',
  Note: '',
  IsCommentLock: false,
  StrokeColor: 'rgba(255,0,0,1)',
  FillColor: 'rgba(255,255,255,0)',
  Opacity: 1,
  Bounds: {
    X: 591,
    Y: 104,
    Width: 100,
    Height: 144,
    Location: { X: 591, Y: 104 },
    Size: { IsEmpty: false, Width: 100, Height: 144 },
    Left: 591,
    Top: 104,
    Right: 691,
    Bottom: 248,
  },
  Thickness: 1,

```

```

BorderStyle: 'Solid',
BorderDashArray: 0,
RotateAngle: 'RotateAngle0',
IsCloudShape: false,
CloudIntensity: 0,
RectangleDifference: null,
VertexPoints: null,
LineHeadStart: null,
LineHeadEnd: null,
IsLocked: false,
AnnotName: '317e203f-e078-4ca1-2125-b86a806dbea5',
Comments: null,
State: '',
StateModel: '',
AnnotType: 'shape',
EnableShapeLabel: false,
LabelContent: null,
LabelFillColor: null,
LabelBorderColor: null,
FontColor: null,
FontSize: 0,
CustomData: null,
LabelBounds: {
  X: 0,
  Y: 0,
  Width: 0,
  Height: 0,
  Location: { X: 0, Y: 0 },
  Size: { IsEmpty: true, Width: 0, Height: 0 },
  Left: 0,
  Top: 0,
  Right: 0,
  Bottom: 0,
},
LabelSettings: null,
AnnotationSettings: {
  minWidth: 0,
  maxWidth: 0,
  minHeight: 0,
  maxHeight: 0,
  isLock: false,
  isPrint: true,
},
AllowedInteractions: ['None'],
IsPrint: true,
ExistingCustomData: null,
},
],
},
},
});
}
}

```

SERVER-BACKED

```

import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService],
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  //The service must be running in the local
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  //Local file path
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  OnImportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.importAnnotation({
      pdfAnnotation: {
        '0': {
          shapeAnnotation: [
            {
              ShapeAnnotationType: 'Square',
              Author: 'Guest',
              AnnotationSelectorSettings: {
                selectionBorderColor: '',
                resizerBorderColor: 'black',
                resizerFillColor: '#FF4081',
                resizerSize: 8,
                selectionBorderThickness: 1,
                resizerShape: 'Square',
                selectorLineDashArray: [],
                resizerLocation: 3,
                resizerCursorType: null,
              },
              ModifiedDate: '5/19/2021, 11:44:51 AM',
              Subject: 'Rectangle',
              Note: '',
            }
          ]
        }
      }
    });
  }
}

```



```

IsCommentLock: false,
StrokeColor: 'rgba(255,0,0,1)',
FillColor: 'rgba(255,255,255,0)',
Opacity: 1,
Bounds: {
  X: 154,
  Y: 71,
  Width: 299,
  Height: 173,
  Location: { X: 154, Y: 71 },
  Size: { IsEmpty: false, Width: 299, Height: 173 },
  Left: 154,
  Top: 71,
  Right: 453,
  Bottom: 244,
},
Thickness: 1,
BorderStyle: 'Solid',
BorderDashArray: 0,
RotateAngle: 'RotateAngle0',
IsCloudShape: false,
CloudIntensity: 0,
RectangleDifference: null,
VertexPoints: null,
LineHeadStart: null,
LineHeadEnd: null,
IsLocked: false,
AnnotName: 'e9ba2384-a202-45ff-0146-444fa0424dc5',
Comments: null,
State: '',
StateModel: '',
AnnotType: 'shape',
EnableShapeLabel: false,
LabelContent: null,
LabelFillColor: null,
LabelBorderColor: null,
FontColor: null,
FontSize: 0,
CustomData: null,
LabelBounds: {
  X: 0,
  Y: 0,
  Width: 0,
  Height: 0,
  Location: { X: 0, Y: 0 },
  Size: { IsEmpty: true, Width: 0, Height: 0 },
  Left: 0,
  Top: 0,
  Right: 0,
  Bottom: 0,
},
LabelSettings: null,
AnnotationSettings: {
  minWidth: 0,
  maxWidth: 0,
  minHeight: 0,
  maxHeight: 0,

```

```

isLock: false,
isPrint: true,
},
AllowedInteractions: ['None'],
IsPrint: true,
ExistingCustomData: null,
},
{
ShapeAnnotationType: 'Circle',
Author: 'Guest',
AnnotationSelectorSettings: {
selectionBorderColor: '',
resizerBorderColor: 'black',
resizerFillColor: '#FF4081',
resizerSize: 8,
selectionBorderThickness: 1,
resizerShape: 'Square',
selectorLineDashArray: [],
resizerLocation: 3,
resizerCursorType: null,
},
ModifiedDate: '5/19/2021, 11:44:53 AM',
Subject: 'Circle',
Note: '',
IsCommentLock: false,
StrokeColor: 'rgba(255,0,0,1)',
FillColor: 'rgba(255,255,255,0)',
Opacity: 1,
Bounds: {
X: 591,
Y: 104,
Width: 100,
Height: 144,
Location: { X: 591, Y: 104 },
Size: { IsEmpty: false, Width: 100, Height: 144 },
Left: 591,
Top: 104,
Right: 691,
Bottom: 248,
},
Thickness: 1,
BorderStyle: 'Solid',
BorderDashArray: 0,
RotateAngle: 'RotateAngle0',
IsCloudShape: false,
CloudIntensity: 0,
RectangleDifference: null,
VertexPoints: null,
LineHeadStart: null,
LineHeadEnd: null,
IsLocked: false,
AnnotName: '317e203f-e078-4ca1-2125-b86a806dbea5',
Comments: null,
State: '',
StateModel: '',
AnnotType: 'shape',
EnableShapeLabel: false,

```

```

LabelContent: null,
LabelFillColor: null,
LabelBorderColor: null,
FontColor: null,
FontSize: 0,
CustomData: null,
LabelBounds: {
  X: 0,
  Y: 0,
  Width: 0,
  Height: 0,
  Location: { X: 0, Y: 0 },
  Size: { IsEmpty: true, Width: 0, Height: 0 },
  Left: 0,
  Top: 0,
  Right: 0,
  Bottom: 0,
},
LabelSettings: null,
AnnotationSettings: {
  minWidth: 0,
  maxWidth: 0,
  minHeight: 0,
  maxHeight: 0,
  isLock: false,
  isPrint: true,
},
AllowedInteractions: ['None'],
IsPrint: true,
ExistingCustomData: null,
},
],
},
},
});
}
}

```

Refer to the following code snippet to import annotations from a JSON file.

STANDALONE

```

import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="onImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"

```

```
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService,TextSelectionService, PrintService,
AnnotationService],
})
export class AppComponent implements OnInit {
@ViewChild('pdfviewer')
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
onImportAnnotationsClick() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.importAnnotation('PDF_Succinctly.json', AnnotationDataFormat.Json);
}
}
```

SERVER-BACKED

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-root',
// specifies the template string for the PDF Viewer component
template: `<button (click)="onImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService,TextSelectionService, PrintService,
AnnotationService],
})
export class AppComponent implements OnInit {
@ViewChild('pdfviewer')
public service: string = 'https://localhost:44347/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
onImportAnnotationsClick() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.importAnnotation('PDF_Succinctly.json', AnnotationDataFormat.Json);
}
}
```

```
}

```

Run the [web service](#) and then the angular code. Also note that, the JSON file for importing the annotation should be placed in the location as specified in the GetDocumentPath method of the PdfViewerController.

Refer to the following code snippet to import annotations from an XPDF file.

STANDALONE

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService ]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  OnImportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfviewer.importAnnotation('PDF_Succinctly.xfdf', AnnotationDataFormat.xfdf);
  }
}
```

SERVER-BACKED

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',

```

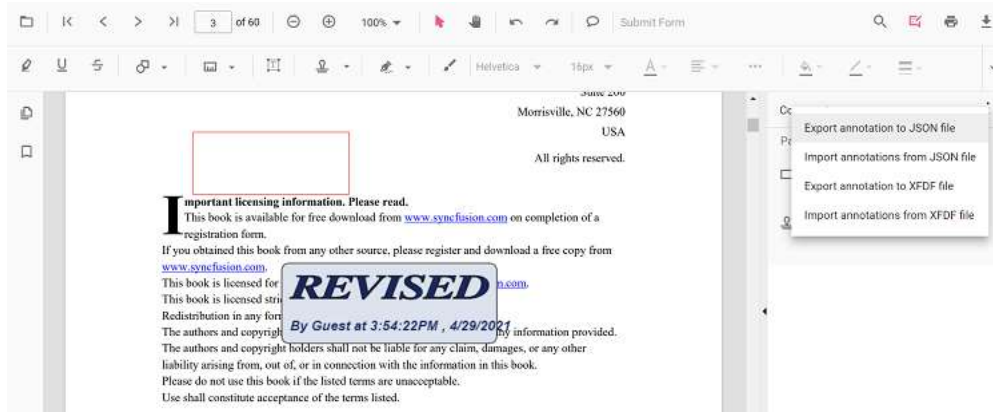
```
// specifies the template string for the PDF Viewer component
template: `<button (click)="OnImportAnnotationsClick()">Import
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService ]
})
export class AppComponent implements OnInit {
@ViewChild('pdfviewer')
public pdfviewerControl: PdfViewerComponent;
public service: string = 'https://localhost:44347/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
OnImportAnnotationsClick() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.importAnnotation('PDF_Succinctly.xfdf', AnnotationDataFormat.xfdf);
}
}
```

Run the [web service](#) and then the angular code. Also note that, the XFDF file for importing the annotation should be placed in the location as specified in the GetDocumentPath method of the PdfViewerController.

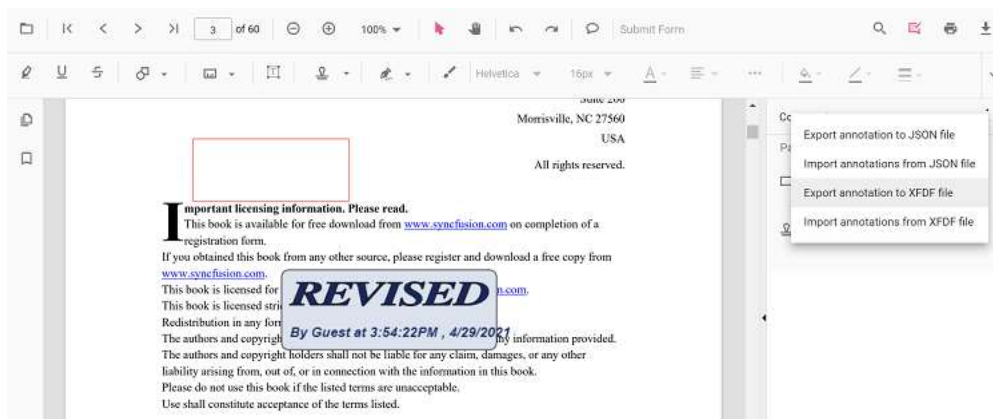
Exporting annotation from the PDF document

The PDF Viewer control provides the support to export the annotations as JSON file or JSON object and XFDF file using annotation toolbar.

- Click the Add or Edit annotation button in the PDF Viewer toolbar.
- The annotation toolbar will appear.
- Click the Comment Panel button in the annotation toolbar.
- The comments panel will displayed.
- Click the **More Option** button in the comment panel container.
- Select the Export annotation to JSON file option to export annotations to a JSON file.



- Select the Export annotation to XPDF file option to export annotations to an XPDF file.



Export annotations will be in the disabled state when the loaded PDF document does not contain any annotations.

[Exporting annotation using PDF Viewer API](#)

You can export annotations as JSON file in code behind like the following code snippet.

STANDALONE

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnExportAnnotationsClick()">Export
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[exportAnnotationFileName]='exportAnnotationFileName'
style="height:640px;display:block">
```

```

</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public exportAnnotationFileName: string = 'ExportedAnnotations.json';
  OnExportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.exportAnnotation(AnnotationDataFormat.Json);
  }
}

```

SERVER-BACKED

```

import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnExportAnnotationsClick()">Export
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[exportAnnotationFileName]='exportAnnotationFileName'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public exportAnnotationFileName: string = 'ExportedAnnotations.json';

```



```
OnExportAnnotationsClick() {
  var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
  pdfviewer.exportAnnotation(AnnotationDataFormat.Json);
}
}
```

Refer to the following code snippet to export annotations as Xfdf file.

STANDALONE

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnExportAnnotationsClick()">Export
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[exportAnnotationFileName]='exportAnnotationFileName'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public exportAnnotationFileName: string = 'ExportedAnnotations.xfdf';
  OnExportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.exportAnnotation(AnnotationDataFormat.Xfdf);
  }
}
```

SERVER-BACKED

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, AnnotationDataFormat
} from '@syncfusion/ej2-angular-pdfviewer';
```

```

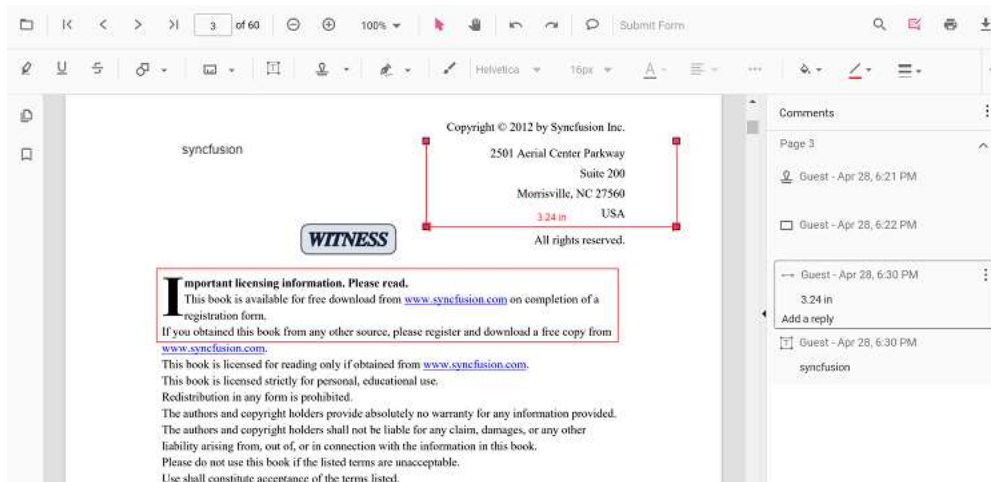
@Component ({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<button (click)="OnExportAnnotationsClick()">Export
Annotations</button>
<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[exportAnnotationFileName]='exportAnnotationFileName'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public exportAnnotationFileName: string = 'ExportedAnnotations.xfdf';
  OnExportAnnotationsClick() {
    var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
    pdfviewer.exportAnnotation(AnnotationDataFormat.Xfdf);
  }
}

```

Comments

The PDF Viewer control provides options to add, edit, and delete the comments to the following annotation in the PDF documents:

- Shape annotation
- Stamp annotation
- Sticky note annotation
- Measurement annotation
- Text markup annotation
- Free text annotation
- Ink annotation



Adding a comment to the annotation

Annotation comment, comment replies, and status can be added to the PDF document using the comment panel.

Comment panel

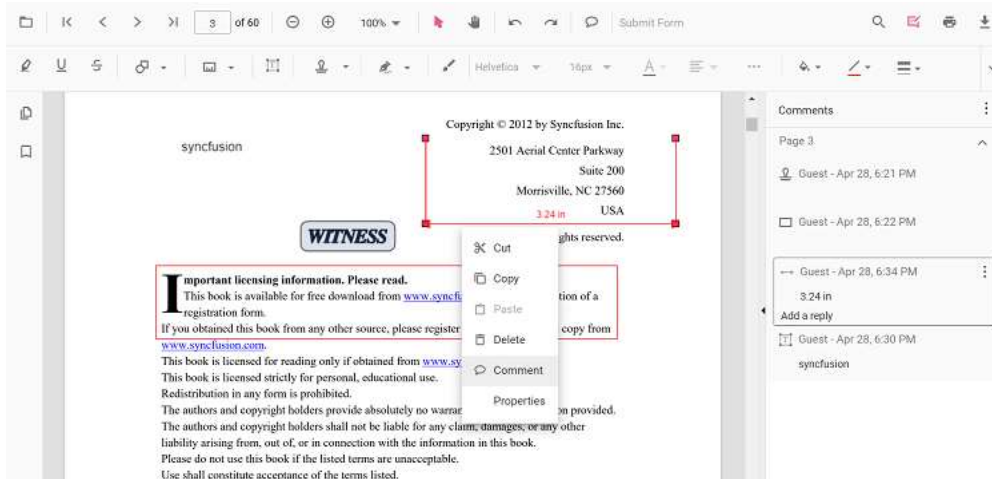
Annotation comments can be added to the PDF using the comment panel. The comment panel can be opened in the following ways:

1. Using the annotation menu
 - Click the Edit Annotation button in the PDF Viewer toolbar. A toolbar appears below it.
 - Click the Comment Panel button. A comment panel will appear.
2. Using Context menu
 - Select annotation in the PDF document and right-click it.
 - Select the comment option in the context menu that appears.
3. Using the Mouse click
 - Select annotation in the PDF document and double click it, a comment panel will appear.

If the comment panel is already in the open state, you can select the annotations and add annotation comments using the comment panel.

Adding comments

- Select annotation in the PDF document and click it.
- The selected annotation comment container is highlighted in the comment panel.
- Now, you can add comment and comment replies using the comment panel.

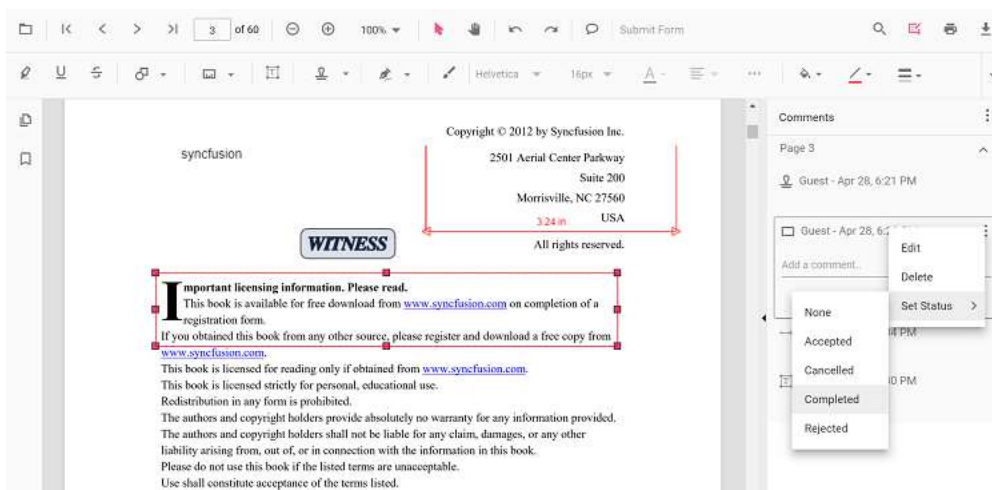


Adding Comment Replies

- The PDF Viewer control provides an option to add multiple replies to the comment.
- After adding the annotation comment, you can add a reply to the comment.

Adding Comment or Reply Status

- Select the Annotation Comments in the comment panel.
- Click the more options button showing in the Comments or reply container.
- Select the Set Status option in the context menu that appears.
- Select the status of the annotation comment in the context menu that appears.



Editing the comments and comments replies of the annotations

The comment, comment replies, and status of the annotation can be edited using the comment panel.

Editing the Comment or Comment Replies

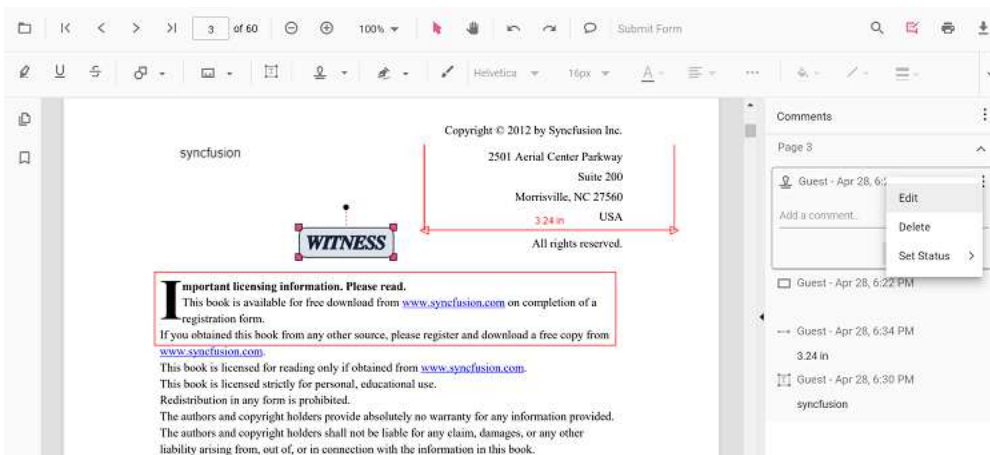
The annotation comment and comment replies can be edited in the following ways:

1. Using the Context menu
 - Select the Annotation Comments in the comment panel.

- Click the More options button showing in the Comments or reply container.
 - Select the Edit option in the context menu that appears.
 - Now, an editable text box appears. You can change the content of the annotation comment or comment reply.
2. Using the Mouse Click
- Select the annotation comments in the comment panel.
 - Double click the comment or comment reply content.
 - Now, an editable text box appears. You can change the content of the annotation comment or comment reply.

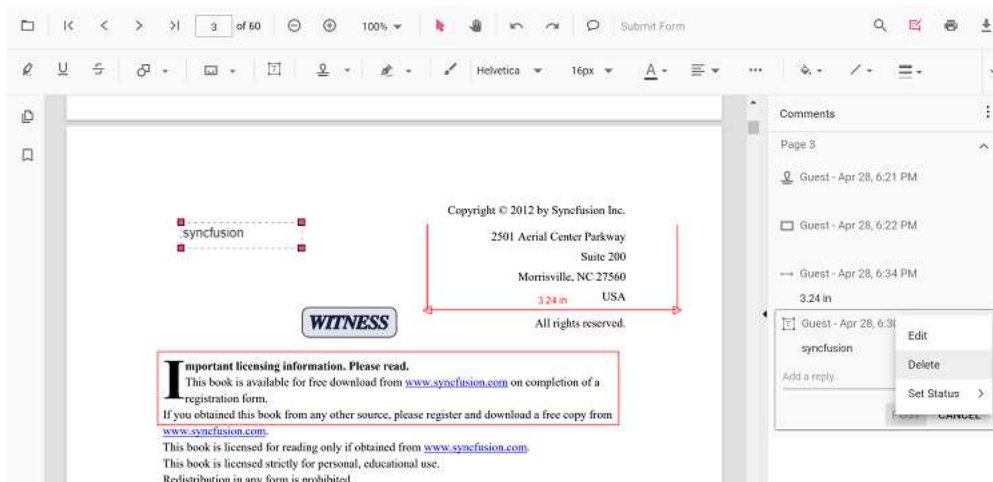
Editing Comment or Reply Status

- Select the Annotation Comments in the comment panel.
- Click the more options button showing in the Comments or reply container.
- Select the Set Status option in the context menu that appears.
- Select the status of the annotation comment in the context menu that appears.
- Status 'None' is the default state. If the status is set to 'None,' the comments or reply does not appear.



Delete Comment or Comment Replies

- Select the Annotation Comments in the comment panel.
- Click the more options button shown in the Comments or reply container.
- Select the Delete option in the context menu that appears.



The annotation will be deleted on deleting the comment using comment panel.

How to check the comments added by the user

The comments added to the PDF document can be viewed by using the `comments` property of the annotation.

Refer to the following code to check the comments added in the PDF document using a button click event.

STANDALONE

```

<html>
<button (click)="checkComments()">Check the Comments</button>
<!--Render PDF Viewer component-->
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</html>

```

SERVER-BACKED

```

<html>
<button (click)="checkComments()">Check the Comments</button>
<!--Render PDF Viewer component-->
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</html>

```

``typescript`

`//Method to check the comments added in the PDF document.`

`checkComments(){`

`var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];`

```

var annotationCollections = pdfviewer.annotationCollection;
for (var x = 0; x < annotationCollections.length; x++) {
  //Prints the annotation id in the console window.
  console.log(annotationCollections[x].annotationId);
  var comments = annotationCollections[x].comments;
  for (var y = 0; y < comments.length; y++) {
    var comment = comments[y];
    //Prints the PDF document's comments in the console window.
    console.log("comment" + "[" + y + "]" + ":" + comment.note);
  }
  var note = annotationCollections[x].note;
  console.log("note : " + note);
}

```

[View sample in GitHub](#)

Handwritten Signature

The PDF Viewer control supports adding the handwritten signatures to a PDF document. The handwritten signature reduces the paperwork of reviewing the content and verifies it digitally.

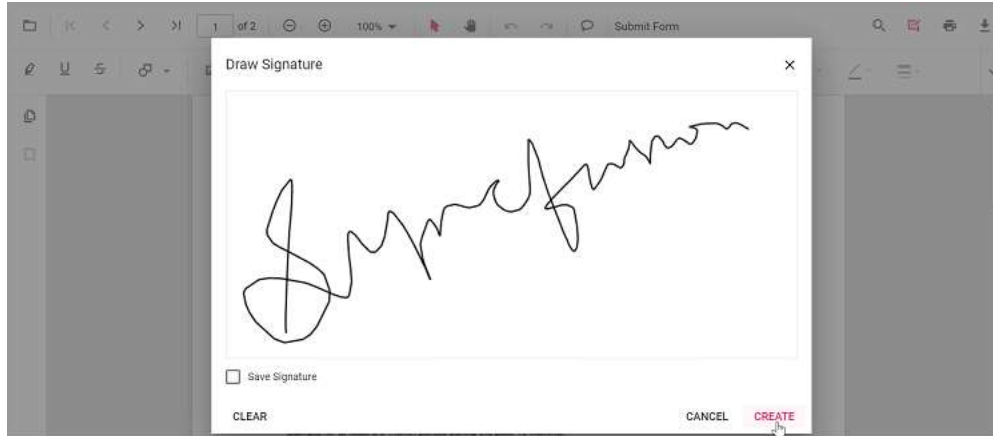
Adding a handwritten signature to the PDF document

The handwritten signature can be added to the PDF document using the annotation toolbar.

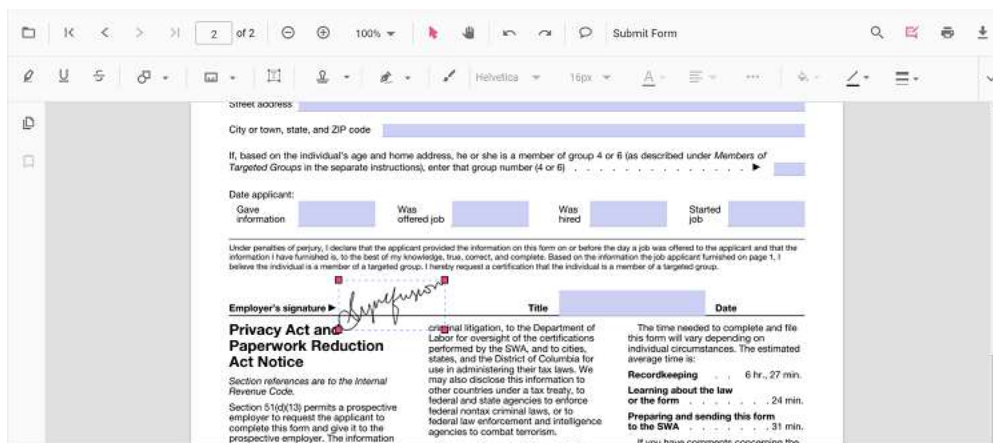
- Click the **Edit Annotation** button in the PDF Viewer toolbar. A toolbar appears below it.
- Select the **Handwritten Signature** button in the annotation toolbar. The signature panel will appear.

The screenshot displays the PDF Viewer application. At the top, a toolbar includes navigation and editing tools. A 'Draw Signature' toolbar is visible, showing various drawing tools like a pen, highlighter, and eraser. Below this, a sample PDF form is shown. The form is titled '8850 Pre-Screening Notice and Certification Request for the Work Opportunity Credit' and is from the Department of the Treasury, Internal Revenue Service. It contains several text input fields for personal information, including name, social security number, address, city, state, ZIP code, county, and telephone number. There are also checkboxes for certification and statements regarding family assistance and SNAP benefits.

- Draw the signature in the signature panel.

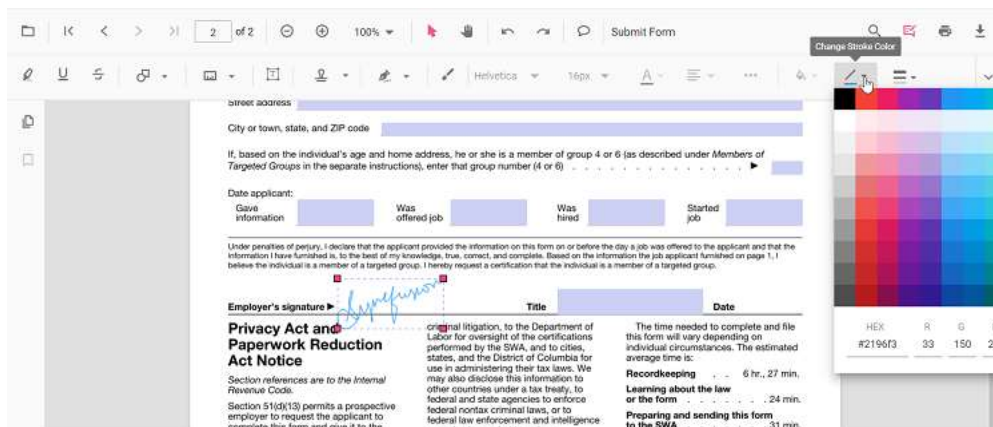


- Then, click the **Create** button and move the signature using the mouse and place them in the desired location.



Editing the properties of handwritten signature

The stroke color, border thickness, and opacity of the handwritten signature can be edited using the edit stroke color tool, edit thickness tool, and edit opacity tool in the annotation toolbar.



Interaction Mode in Angular PDF Viewer component

The PDF Viewer provides interaction mode for easy interaction with the loaded PDF document. Selection mode and panning mode are the two interactions modes.

Selection mode

In this mode, the text selection can be performed in the PDF document loaded in PDF Viewer. The panning and scrolling of the pages by touch cannot be performed in this mode. It allows users to select and copy text from the PDF files. This is helpful for copying and sharing text content. You can enable/disable the text selection using the following code snippet.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
enableTextSelection='true'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
enableTextSelection='true'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

```

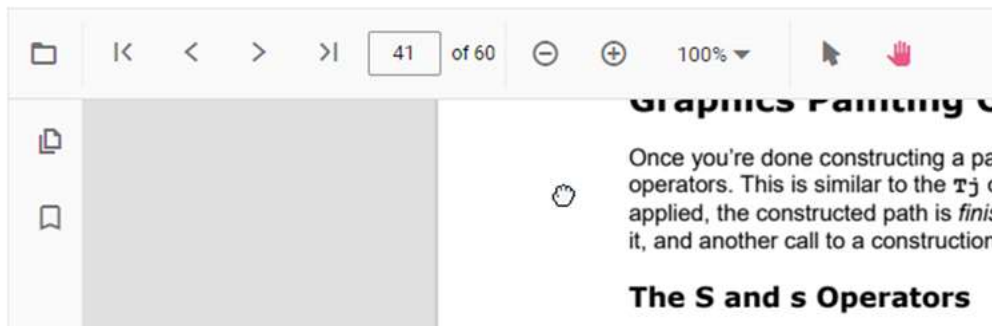
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
}))
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

```



Panning Mode

In this mode, the panning and scrolling of the pages by touch can be performed in the PDF document loaded in the PDF Viewer, but the text selection cannot be performed.



You can switch the interaction mode of PDF Viewer by using the following code snippet.,

STANDALONE

```

@Component ({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
enableTextSelection='true'
[documentPath]='document'
[interactionMode]='interaction'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,

```

```

AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public interaction = 'Pan';
}

```

SERVER-BACKED

```

@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
enableTextSelection='true'
[documentPath]='document'
[interactionMode]='interaction'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public interaction = 'Pan';
}

```

See also

- [Toolbar items](#)
- [Feature Modules](#)

Form Designer

Create form fields programmatically

The PDF Viewer control provides the option to add, edit and delete the Form Fields. The Form Fields type supported by the PDF Viewer Control are:

- Textbox
- Password
- CheckBox
- RadioButton
- ListBox

- DropDown
- SignatureField
- InitialField

Add a form field to PDF document programmatically

Using addFormField method, the form fields can be added to the PDF document programmatically. We need to pass two parameters in this method. They are Form Field Type and Properties of Form Field Type. To add form field programmatically, Use the following code.

APP.COMPONENT.TS

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService,
ToolBarService, NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService,
FormDesignerService, FormFieldsService, TextFieldSettings,
SignatureFieldSettings, InitialFieldSettings,
CheckBoxFieldSettings, RadioButtonFieldSettings } from '@syncfusion/ej2-
angular-pdfviewer';
@Component({
selector: 'app-container',
// Specifies the template string for the PDF Viewer component.
template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
#pdfviewer
[documentPath]='document'
[resourceUrl]='resource'
(documentLoad)='documentLoaded($event)'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolBarService,
NavigationService, TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
@ViewChild('pdfviewer')
public pdfviewerControl?: PdfViewerComponent;
public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
documentLoaded(e: any): void {
this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", {name:
'First Name', bounds: { X: 146, Y: 229, Width: 150, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", { name:
"Middle Name", bounds: { X: 338, Y: 229, Width: 150, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name:
'Last Name',bounds: { X: 530, Y: 229, Width: 150, Height: 24 },} as
TextFieldSettings);
```

```

this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
{bounds: { X: 148, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
false,} as RadioButtonFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
{bounds: { X: 292, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
false,} as RadioButtonFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
Month',bounds: { X: 146, Y: 320, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
Date',bounds: { X: 193, Y: 320, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
Year',bounds: { X: 242, Y: 320, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
'Agree',bounds: { X: 148, Y: 408, Width: 200, Height: 43 },} as
InitialFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
'Do Not Agree',bounds: { X: 148, Y: 466, Width: 200, Height: 43 },} as
InitialFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Text Message',bounds: { X: 56, Y: 664, Width: 20, Height: 20 },isChecked:
false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Email',bounds: { X: 242, Y: 664, Width: 20, Height: 20 },isChecked: false,}
as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Appointment Reminder',bounds: { X: 56, Y: 740, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Request for Customerservice',bounds: { X: 56, Y: 778, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Information Billing',bounds: { X: 290, Y: 740, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
Email',bounds: { X: 146, Y: 850, Width: 200, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
Phone',bounds: { X: 482, Y: 850, Width: 200, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('SignatureField',
{name: 'Sign',bounds: { X: 57, Y: 923, Width: 200, Height: 43 },} as
SignatureFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Month',bounds: { X: 386, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Date',bounds: { X: 434, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Year',bounds: { X: 482, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
}
ngOnInit(): void {
}

```

```
}

```

MAIN.TS

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

Note: To set up the **server-backed PDF Viewer**,

Add the below `serviceUrl` in the `app.component.ts` file

```
public service: string = 'https://services.syncfusion.com/angular/production/api/pdfviewer';
```

Within the template, configure the PDF Viewer by adding the `[serviceUrl]='service'` attribute inside the `div` element.

Edit/Update form field programmatically

Using `updateFormField` method, Form Field can be updated programmatically. We should get the Form Field object/Id from `FormFieldCollections` property that you would like to edit and pass it as a parameter to `updateFormField` method. The second parameter should be the properties that you would like to update for Form Field programmatically. We have updated the value and background Color properties of Textbox Form Field.

APP.COMPONENT.TS

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService,
ToolbarService, NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService,
FormDesignerService, FormFieldsService, TextFieldSettings,
SignatureFieldSettings, InitialFieldSettings,
CheckBoxFieldSettings, RadioButtonFieldSettings } from '@syncfusion/ej2-
angular-pdfviewer';
@Component({
  selector: 'app-container',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer
id="pdfViewer"
#pdfviewer
[documentPath]='document'
[resourceUrl]='resource'
(documentLoad)='documentLoaded($event)'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
```

```

export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl?: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
  designer.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
  pdfviewer-lib";
  documentLoaded(e: any): void {
    this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", {name:
    'First Name', bounds: { X: 146, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", { name:
    "Middle Name", bounds: { X: 338, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name:
    'Last Name',bounds: { X: 530, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
    {bounds: { X: 148, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
    false,} as RadioButtonFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
    {bounds: { X: 292, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
    false,} as RadioButtonFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Month',bounds: { X: 146, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Date',bounds: { X: 193, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Year',bounds: { X: 242, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
    'Agree',bounds: { X: 148, Y: 408, Width: 200, Height: 43 },} as
    InitialFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
    'Do Not Agree',bounds: { X: 148, Y: 466, Width: 200, Height: 43 },} as
    InitialFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
    'Text Message',bounds: { X: 56, Y: 664, Width: 20, Height: 20 },isChecked:
    false,} as CheckBoxFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
    'Email',bounds: { X: 242, Y: 664, Width: 20, Height: 20 },isChecked: false,}
    as CheckBoxFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
    'Appointment Reminder',bounds: { X: 56, Y: 740, Width: 20, Height: 20
    },isChecked: false,} as CheckBoxFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
    'Request for Customerservice',bounds: { X: 56, Y: 778, Width: 20, Height: 20
    },isChecked: false,} as CheckBoxFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
    'Information Billing',bounds: { X: 290, Y: 740, Width: 20, Height: 20
    },isChecked: false,} as CheckBoxFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
    Email',bounds: { X: 146, Y: 850, Width: 200, Height: 24 },} as
    TextFieldSettings);
  }
}

```

```

this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
Phone',bounds: { X: 482, Y: 850, Width: 200, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('SignatureField',
{name: 'Sign',bounds: { X: 57, Y: 923, Width: 200, Height: 43 },} as
SignatureFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Month',bounds: { X: 386, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Date',bounds: { X: 434, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Year',bounds: { X: 482, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.updateFormField(this.pdfviewerContr
ol?.formFieldCollections[0], { backgroundColor: 'red' } as
TextFieldSettings);
}
ngOnInit(): void {
}
}

```

MAIN.TS

```

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

```

Note: To set up the **server-backed PDF Viewer**,

Add the below `serviceUrl` in the `app.component.ts` file

```
public service: string = 'https://services.syncfusion.com/angular/production/api/pdfviewer';
```

Within the template, configure the PDF Viewer by adding the `[serviceUrl]='service'` attribute inside the `div` element.

Delete form field programmatically

Using `deleteFormField` method, the form field can be deleted programmatically. We should retrieve the Form Field object/Id from `FormFieldCollections` property that you would like to delete and pass it as a parameter to `deleteFormField` method. To delete a Form Field programmatically, use the following code.

APP.COMPONENT.TS

```

import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService,
ToolbarService, NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService,
FormDesignerService, FormFieldsService, TextFieldSettings,
SignatureFieldSettings, InitialFieldSettings,
CheckBoxFieldSettings, RadioButtonFieldSettings } from '@syncfusion/ej2-
angular-pdfviewer';

```



```

@Component ({
  selector: 'app-container',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
    <ejs-pdfviewer
      id="pdfViewer"
      #pdfviewer
      [documentPath]='document'
      [resourceUrl]='resource'
      (documentLoad)='documentLoaded($event)'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
    ThumbnailViewService, ToolbarService,
    NavigationService, TextSearchService, TextSelectionService, PrintService,
    AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl?: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
  designer.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
  pdfviewer-lib";
  documentLoaded(e: any): void {
    this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", {name:
    'First Name', bounds: { X: 146, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField("Textbox", { name:
    "Middle Name", bounds: { X: 338, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name:
    'Last Name',bounds: { X: 530, Y: 229, Width: 150, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
    {bounds: { X: 148, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
    false,} as RadioButtonFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('RadioButton',
    {bounds: { X: 292, Y: 289, Width: 18, Height: 18 },name: 'Gender',isSelected:
    false,} as RadioButtonFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Month',bounds: { X: 146, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Date',bounds: { X: 193, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOB
    Year',bounds: { X: 242, Y: 320, Width: 35, Height: 24 },} as
    TextFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
    'Agree',bounds: { X: 148, Y: 408, Width: 200, Height: 43 },} as
    InitialFieldSettings);
    this.pdfviewerControl?.formDesignerModule.addFormField('InitialField', {name:
    'Do Not Agree',bounds: { X: 148, Y: 466, Width: 200, Height: 43 },} as
    InitialFieldSettings);
  }
}

```

```

this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Text Message',bounds: { X: 56, Y: 664, Width: 20, Height: 20 },isChecked:
false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Email',bounds: { X: 242, Y: 664, Width: 20, Height: 20 },isChecked: false,}
as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Appointment Reminder',bounds: { X: 56, Y: 740, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Request for Customerservice',bounds: { X: 56, Y: 778, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('CheckBox', {name:
'Information Billing',bounds: { X: 290, Y: 740, Width: 20, Height: 20
},isChecked: false,} as CheckBoxFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
Email',bounds: { X: 146, Y: 850, Width: 200, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'My
Phone',bounds: { X: 482, Y: 850, Width: 200, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('SignatureField',
{name: 'Sign',bounds: { X: 57, Y: 923, Width: 200, Height: 43 },} as
SignatureFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Month',bounds: { X: 386, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Date',bounds: { X: 434, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.addFormField('Textbox', {name: 'DOS
Year',bounds: { X: 482, Y: 923, Width: 35, Height: 24 },} as
TextFieldSettings);
this.pdfviewerControl?.formDesignerModule.deleteFormField(this.pdfviewerContr
ol.formFieldCollections[0]);
}
ngOnInit(): void {
}
}

```

MAIN.TS

```

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

```

Note: To set up the **server-backed PDF Viewer**,

Add the below `serviceUrl` in the `app.component.ts` file

```
public service: string = 'https://services.syncfusion.com/angular/production/api/pdfviewer';
```

Within the template, configure the PDF Viewer by adding the `[serviceUrl]='service'` attribute inside the `div` element.

The following code illustrates how to delete a signature from the signature field using the `retrieveFormFields` and `clearFormFields` APIs.

STANDALONE

```
<!--Method to remove signature-->
<button (click)="removeSignature()">Remove Signature</button>
<!--Render PDF Viewer component-->
<ejs-pdfviewer id="pdfViewer"
[documentPath]="document"
style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<!--Method to remove signature-->
<button (click)="removeSignature()">Remove Signature</button>
<!--Render PDF Viewer component-->
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]="service"
[documentPath]="document"
style="height:640px;display:block">
</ejs-pdfviewer>
```

`typescript

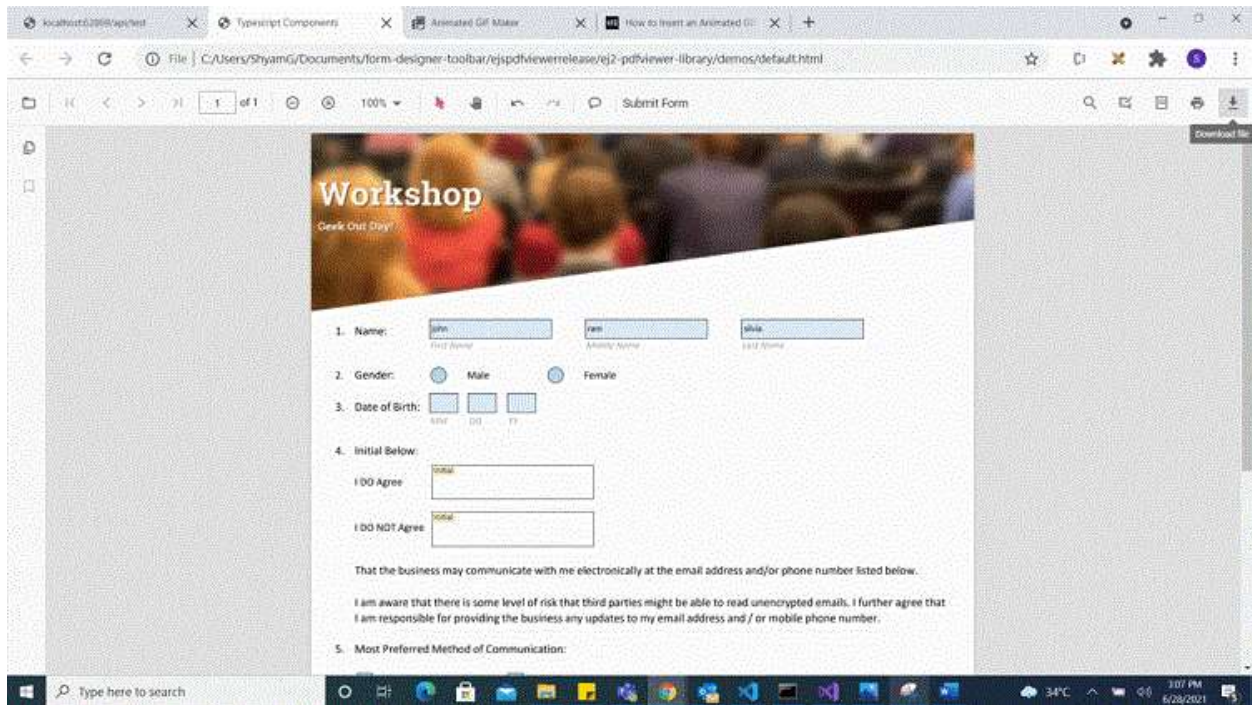
//Event triggers while clicking the Remove Signature button.

```
removeSignature() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
var forms = viewer.retrieveFormFields();
//API to remove a signature from the signature field.
viewer.clearFormFields(forms[8]);
}
`
```

[View sample in GitHub](#)

Saving the form fields

When the download icon is selected on the toolbar, the Form Fields will be saved in the PDF document and this action will not affect the original document. Refer the below GIF for further reference.

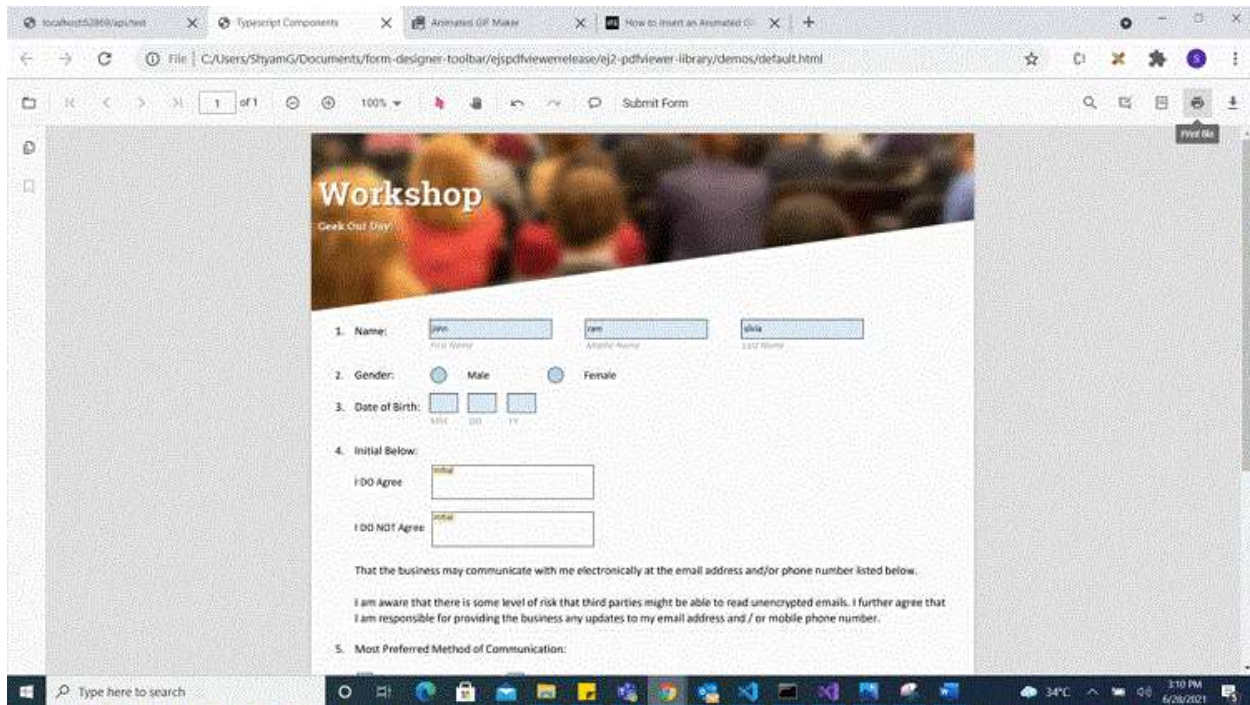


You can invoke download action using following code snippet.

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.download();
}
</script>
`
```

Printing the form fields

When the print icon is selected on the toolbar, the PDF document will be printed along with the Form Fields added to the pages and this action will not affect the original document. Refer the below GIF for further reference.

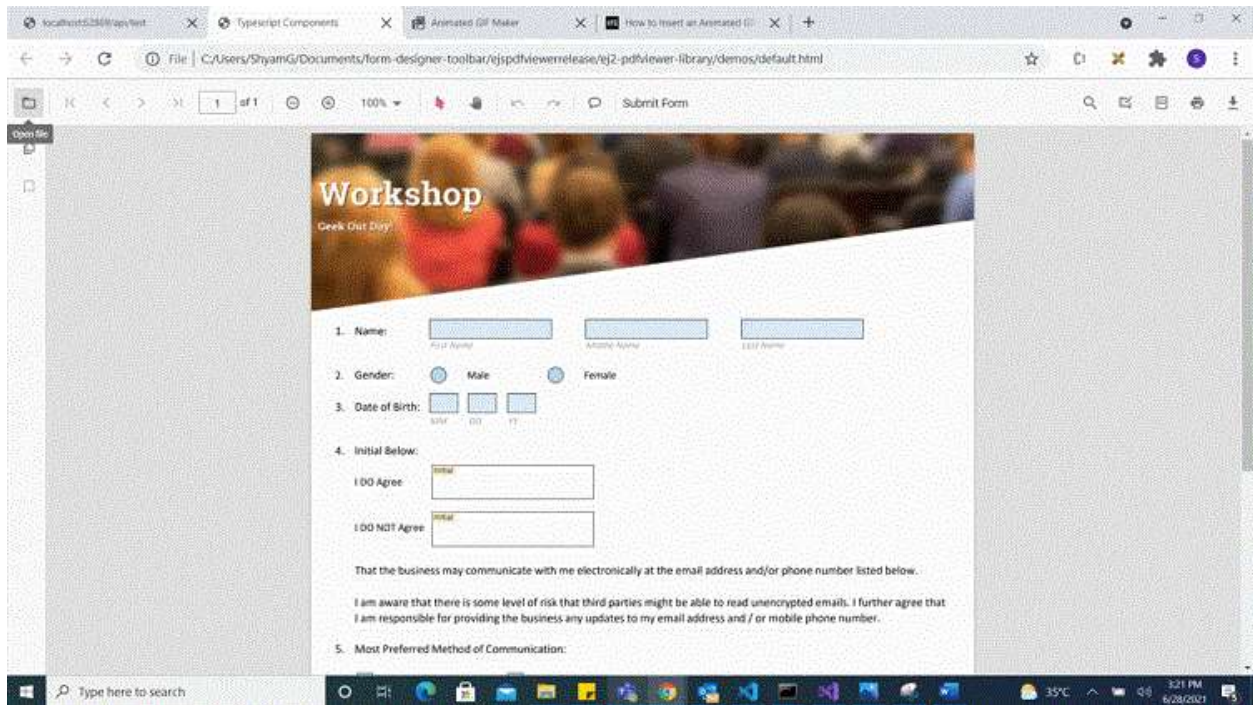


You can invoke print action using the following code snippet.,

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.print.print();
}
</script>
`
```

Open the existing PDF document

We can open the already saved PDF document contains Form Fields in it by clicking the open icon in the toolbar. Refer the below GIF for further reference.



Validate form fields

The form fields in the PDF Document will be validated when the `enableFormFieldsValidation` is set to `true` and hook the `validateFormFields`. The `validateFormFields` will be triggered when the PDF document is downloaded or printed with the non-filled form fields. The non-filled fields will be obtained in the `nonFillableFields` property of the event arguments of `validateFormFields`.

Add the following code snippet to validate the form fields,

STANDALONE

```

```typescript
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService,
FormFieldsService, LoadEventArgs, TextFieldSettings
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
 selector: 'app-root',
 // Specifies the template string for the PDF Viewer component.
 template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
#pdfviewer
[documentPath]='document'
[enableFormFieldsValidation]=true
(validateFormFields)='validateFormFields($event)'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
 providers: [LinkAnnotationService, BookmarkViewService,
MagnificationService,

```

```
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
 @ViewChild('pdfviewer')
 public pdfviewerControl: PdfViewerComponent;
 public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
 public validateFormFields(e: ValidateFormFieldsArgs): void {
 this.e.nonFillableFields;
 }
}
....
```

### SERVER-BACKED

```
````typescript
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService,
FormFieldsService, LoadEventArgs, TextFieldSettings
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
#pdfviewer [serviceUrl]='service'
[documentPath]='document'
[enableFormFieldsValidation]=true
(validateFormFields)='validateFormFields($event)'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
  public validateFormFields(e: ValidateFormFieldsArgs): void {
    this.e.nonFillableFields;
  }
}
....
```

Export and import form fields

The PDF Viewer control provides the support to export and import the form field data in the following formats using the methods `importFormFields`, `exportFormFields`, `exportFormFieldsAsObject`.

- FDF
- XFDF
- JSON
- XML

Export and import as FDF

Using the `exportFormFields` method, the form field data can be exported in the specified data format. This method accepts two parameters:

- The first one must be the destination path for the exported data. If path is not specified, it will ask for the location while exporting.
- The second parameter should be the format type of the form data.

The following code explains how to export and import the form field data as FDF.

```
`html
```

```
<button (click)="OnExportFdf()">Export FDF</button>
```

```
<button (click)="OnImportFdf()">Import FDF</button>
```

```
,
```

```
`typescript
```

```
// Event triggers on the Export FDF button click.
```

```
OnExportFdf() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

```
// Data must be the desired path for the exported document.
```

```
viewer.exportFormFields('Data', FormFieldDataFormat.Fdf);
```

```
}
```

```
// Event triggers on the Import FDF button click.
```

```
OnImportFdf() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

```
// The file for importing the form fields should be placed in the desired location and the path should be provided correctly
```

```
viewer.importFormFields('File', FormFieldDataFormat.Fdf);
```

```
}
```

```
,
```


Export and import as XFDF

The following code explains how to export and import the form field data as XFDF.

```
`html
<button (click)="OnExportXfdf()">Export XFDF</button>
<button (click)="OnImportXfdf()">Import XFDF</button>
`

`typescript
// Event triggers on the Export XFDF button click.
OnExportXfdf() {
  var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  // Data must be the desired path for the exported document.
  viewer.exportFormFields('Data', FormFieldDataFormat.Xfdf);
}
// Event triggers on the Import XFDF button click.
OnImportXfdf() {
  var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  // The file for importing the form fields should be placed in the desired location and the path should be
  // provided correctly
  viewer.importFormFields('File', FormFieldDataFormat.Xfdf);
}
```

Export and import as JSON

The following code explains how to export and import the form field data as JSON.

```
`html
<button (click)="OnExportJson()">Export JSON</button>
<button (click)="OnImportJson()">Import JSON</button>
`

`typescript
// Event triggers on the Export JSON button click.
OnExportJson() {
  var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  // Data must be the desired path for the exported document.
  viewer.exportFormFields('Data', FormFieldDataFormat.Json);
}
```

// Event triggers on the Import JSON button click.

```
OnImportJson() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

// The file for importing the form fields should be placed in the desired location and the path should be provided correctly

```
viewer.importFormFields('File', FormFieldDataFormat.Json);
```

```
}
```

```
,
```

Export and import as Object

The PDF Viewer control supports exporting the form field data as an object, and the exported data will be imported into the current PDF document from the object.

The following code shows how to export the form field data as an object and import the form field data from that object into the current PDF document via a button click.

```
`html
```

```
<button (click)="exportDataAsObject()">Export Object</button>
```

```
<button (click)="importData()">Import Data</button>
```

```
,
```

```
`typescript
```

```
var exportedData;
```

// Event triggers on the Export Object button click.

```
exportDataAsObject() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

// Export the form fields data to an FDF object.

```
exportedData = viewer.exportFormFieldsAsObject(FormFieldDataFormat.Fdf);
```

```
//// Export the form fields data to an XFDF object.
```

```
//exportedData = viewer.exportFormFieldsAsObject(FormFieldDataFormat.Xfdf);
```

```
//// Export the form fields data to an JSON object.
```

```
//exportedData = viewer.exportFormFieldsAsObject(FormFieldDataFormat.Json);
```

```
}
```

// Event triggers on Import Data button click.

```
importData() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

// Import the form fields data from the FDF object into the current PDF document.

```
viewer.importFormFields(exportedData, FormFieldDataFormat.Fdf);
```

```

//// Import the form fields data from the XFDF object into the current PDF document.
//viewer.importFormFields(exportedData, FormFieldDataFormat.Xfdf);
//// Import the form fields data from the JSON object into the current PDF document.
//viewer.importFormFields(exportedData, FormFieldDataFormat.Json);
}
,

```

Export and import as XML

The following code explains how to export and import the form field data as XML.

```

`html
<button (click)="OnExportXml()">Export XML</button>
<button (click)="OnImportXml()">Import XML</button>
,

`typescript
// Event triggers on the Export Xml button click.
OnExportXml() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
// Data must be the desired path for the exported document.
viewer.exportFormFields('Data', FormFieldDataFormat.Xml);
}

// Event triggers on the Import Xml button click.
OnImportXml() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
// The file for importing the form fields should be placed in the desired location and the path should be
provided correctly
viewer.importFormFields('File', FormFieldDataFormat.Xml);
}
,

```

Form field properties

Form field properties in Syncfusion PDF Viewer allow you to customize and interact with form fields embedded within PDF documents. This documentation provides an overview of the form field properties supported by the Syncfusion PDF Viewer and explains how to use them effectively.

- Textbox
- Password
- CheckBox
- RadioButton
- ListBox

- DropDown
- SignatureField
- InitialField

Signature and initial fields settings

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the signature field properties on a button click.

```
`html
<button (click)="updateProperties()">Update Properties</button>
`

`typescript
// Event triggers on the Update Properties button click.
updateProperties() {
  var formField = viewer.retrieveFormFields();
  viewer.formDesignerModule.updateFormField(formField[0], {
    name: 'Initial',
    isReadOnly: true,
    visibility: 'visible',
    isRequired: false,
    isPrint: true,
    tooltip: 'Initial',
    thickness: 4
  });
}
```

The following code example explains how to update the properties of the signature field added to the document from the form designer toolbar.

STANDALONE

```
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]="document"
  [initialFieldSettings] = "initialFieldSettings"
  [signatureFieldSettings] = "signatureFieldSettings"
  style="height:640px;display:block"
> </ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer
```

```
id="pdfViewer"  
[serviceUrl]="service"  
[documentPath]="document"  
[initialFieldSettings] = "initialFieldSettings"  
[signatureFieldSettings] = "signatureFieldSettings"  
style="height:640px;display:block"  
> </ejs-pdfviewer>
```

`typescript

// Properties to customize the signature field settings

public signatureFieldSettings = {

// Set the name of the form field element.

name: 'Signature',

// Specify whether the signature field is in read-only or read-write mode.

isReadOnly: false,

// Set the visibility of the form field.

visibility: 'visible',

// Specify whether the field is mandatory or not.

isRequired: false,

// Specify whether to print the signature field.

isPrint: true,

// Set the text to be displayed as a tooltip.

tooltip: 'Signature',

// Set the thickness of the signature field. To hide the borders, set the value to 0 (zero).

thickness: 4,

// Specify the properties of the signature Dialog Settings in the signature field.

signatureDialogSettings: {

displayMode: DisplayMode.Draw | DisplayMode.Upload | DisplayMode.Text,

hideSaveSignature: false,

},

// Specify the properties of the signature indicator in the signature field.

signatureIndicatorSettings: {

opacity: 1,

backgroundColor: '#237ba2',

height: 50,

fontSize: 15,

```

text: 'Signature Field',
color: 'white'
}
};
`

```



The following code example explains how to update the properties of the initial field added to the document from the form designer toolbar.

```

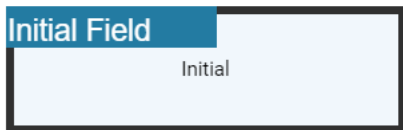
`typescript
// Properties to customize the initial field settings
public initialFieldSettings = {
// Set the name of the form field element.
name: 'Initial',
// Specify whether the initial field is in read-only or read-write mode.
isReadOnly: false,
// Set the visibility of the form field.
visibility: 'visible',
// Specify whether the field is mandatory or not.
isRequired: false,
// Specify whether to print the initial field.
isPrint: true,
// Set the text to be displayed as tooltip.
tooltip: 'Initial',
// Set the thickness of the initial field. To hide the borders, set the value to 0 (zero).
thickness: 4,
// Specify the properties of the initial indicator in the initial field.
initialIndicatorSettings: {
opacity: 1,
backgroundColor: '#237ba2',
height: 50,
fontSize: 15,

```

```

text: 'Initial Field',
color: 'white',
},
// Specify the properties of the initial Dialog Settings in the initial field.
initialDialogSettings: {
displayMode: DisplayMode.Draw | DisplayMode.Upload | DisplayMode.Text,
hideSaveSignature: false,
},
};
`

```



Textbox field settings

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the Textbox field properties on a button click.

```

`html
<button (click)="updateProperties()">Update Properties</button>
`

`typescript
// Event triggers on the Update Properties button click.
updateProperties() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
var formField = viewer.retrieveFormFields();
viewer.formDesignerModule.updateFormField(formField[0], {
name: 'Textbox',
isReadOnly: true,
visibility: 'visible',
isRequired: false,
isPrint: true,
tooltip: 'Textbox',
thickness: 4,

```

```

value:'Textbox',
fontFamily: 'Courier',
fontSize: 10,
fontStyle: 'None',
color: 'black',
borderColor: 'black',
backgroundColor: 'white',
alignment: 'Left',
maxLength: 0,
isMultiline: false,
bounds: { X: 146, Y: 229, Width: 150, Height: 24 }
});
}
`

```

The following code example explains how to update the properties of the Textbox field added to the document from the form designer toolbar.

STANDALONE

```

<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]="document"
  [textFieldSettings] = "textFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>

```

SERVER-BACKED

```

<ejs-pdfviewer
  id="pdfViewer"
  [serviceUrl]="service"
  [documentPath]="document"
  [textFieldSettings] = "textFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>

```

`typescript

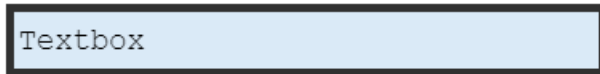
```

// Properties to customize the Textbox field settings
public textFieldSettings = {
// Set the name of the form field element.
name: 'Textbox',
// Specify whether the Textbox field is in read-only or read-write mode.

```



```
isReadOnly: false,  
// Set the visibility of the form field.  
visibility: 'visible',  
// Specify whether the field is mandatory or not.  
isRequired: false,  
// Specify whether to print the Textbox field.  
isPrint: true,  
// Set the text to be displayed as a tooltip.  
tooltip: 'Textbox',  
// Set the thickness of the Textbox field. To hide the borders, set the value to 0 (zero).  
thickness: 4,  
// Set the value of the form field element.  
value:'Textbox',  
// Set the font family of the textbox field.  
fontFamily: 'Courier',  
// Set the font size of the textbox field.  
fontSize: 10,  
// Specify the font style  
fontStyle: 'None',  
// Set the font color of the textbox field.  
color: 'black',  
// Set the border color of the textbox field.  
borderColor: 'black',  
// Set the background color of the textbox field.  
backgroundColor: 'White',  
// Set the alignment of the text.  
alignment: 'Left',  
// Set the maximum character length.  
maxLength: 0,  
// Allows multiline input in the text field. FALSE, by default.  
isMultiline: false  
};  
,
```



Password field settings

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the Password field properties on a button click.

`html

```
<button (click)="updateProperties()">Update Properties</button>
```

,

`typescript

```
// Event triggers on the Update Properties button click.
```

```
updateProperties() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

```
var formField = viewer.retrieveFormFields();
```

```
viewer.formDesignerModule.updateFormField(formField[0], {
```

```
name: 'Password',
```

```
isReadOnly: true,
```

```
visibility: 'visible',
```

```
isRequired: false,
```

```
isPrint: true,
```

```
tooltip: 'Password',
```

```
thickness: 4,
```

```
value: 'Password',
```

```
fontFamily: 'Courier',
```

```
fontSize: 10,
```

```
fontStyle: 'None',
```

```
color: 'black',
```

```
borderColor: 'black',
```

```
backgroundColor: 'white',
```

```
alignment: 'Left',
```

```
maxLength: 0,
```

```
bounds: { X: 148, Y: 229, Width: 150, Height: 24 }
```

```
});
```

```
}
`
```

The following code example explains how to update the properties of the Password field added to the document from the form designer toolbar.

STANDALONE

```
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]="document"
  [passwordFieldSettings] = "passwordFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer
  id="pdfViewer"
  [serviceUrl]="service"
  [documentPath]="document"
  [passwordFieldSettings] = "passwordFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>
```

```
`typescript
```

```
// Properties to customize the Password field settings
```

```
public passwordFieldSettings = {
```

```
// Set the name of the form field element.
```

```
name: 'Password',
```

```
// Specify whether the Password field is in read-only or read-write mode.
```

```
isReadOnly: false,
```

```
// Set the visibility of the form field.
```

```
visibility: 'visible',
```

```
// Specify whether the field is mandatory or not.
```

```
isRequired: false,
```

```
// Specify whether to print the Password field.
```

```
isPrint: true,
```

```
// Set the text to be displayed as a tooltip.
```

```
tooltip: 'Password',
```

```
// Set the thickness of the Password field. To hide the borders, set the value to 0 (zero).
```

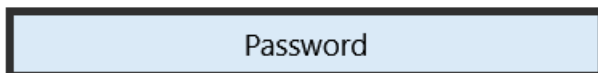
```
thickness: 4,
```

```
// Set the value of the form field element.
```

```

value:'Password',
// Set the font family of the Password field.
fontFamily: 'Courier',
// Set the font size of the Password field.
fontSize: 10,
// Specify the font style
fontStyle: 'None',
// Set the font color of the Password field.
color: 'black',
// Set the border color of the Password field.
borderColor: 'black',
// Set the background color of the Password field.
backgroundColor: 'white',
// Set the alignment of the text.
alignment: 'Left',
// Set the maximum character length.
maxLength: 0,
};

```



CheckBox field settings

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the CheckBox field properties on a button click.

`html

```
<button (click)="updateProperties()">Update Properties</button>
```

,

`typescript

```
// Event triggers on the Update Properties button click.
```

```
updateProperties() {
```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
```

```
var formField = viewer.retrieveFormFields();
```

```
viewer.formDesignerModule.updateFormField(formField[0], {
  name: 'CheckBox',
  isReadOnly: true,
  visibility: 'visible',
  isRequired: false,
  isPrint: true,
  tooltip: 'CheckBox',
  thickness: 4,
  isChecked: true,
  backgroundColor: 'white',
  borderColor: 'black',
  value: 'CheckBox',
});
}
```

The following code example explains how to update the properties of the CheckBox field added to the document from the form designer toolbar.

STANDALONE

```
<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]="document"
  [checkBoxFieldSettings] = "checkBoxFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer
  id="pdfViewer"
  [serviceUrl]="service"
  [documentPath]="document"
  [checkBoxFieldSettings] = "checkBoxFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>
```

`typescript

// Properties to customize the CheckBox field settings

```
public checkBoxFieldSettings = {
```

// Set the name of the form field element.

```
name: 'CheckBox',
```

```
// Specify whether the CheckBox field is in read-only or read-write mode.
readOnly: false,
// Set the visibility of the form field.
visibility: 'visible',
// Specify whether the field is mandatory or not.
isRequired: false,
// Specify whether to print the CheckBox field.
isPrint: true,
// Set the text to be displayed as a tooltip.
tooltip: 'CheckBox',
// Set the thickness of the CheckBox field. To hide the borders, set the value to 0 (zero).
thickness: 4,
// Specifies whether the check box is in checked state or not.
isChecked: true,
// Set the background color of the check box in hexadecimal string format.
backgroundColor: 'white',
// Set the border color of the check box field.
borderColor: 'black',
// Set the value of the form field element.
value: 'CheckBox'
};
`
```



RadioButton field settings

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the RadioButton field properties on a button click.

```
`html
<button (click)="updateProperties()">Update Properties</button>
`
```

```
`typescript
// Event triggers on the Update Properties button click.
updateProperties() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
var formField = viewer.retrieveFormFields();
viewer.formDesignerModule.updateFormField(formField[0], {
name: 'RadioButton',
isReadOnly: true,
visibility: 'visible',
isRequired: false,
isPrint: true,
tooltip: 'RadioButton',
thickness: 4,
isSelected: true,
backgroundColor: 'white',
borderColor: 'black',
value: 'RadioButton'
});
}
`
```

The following code example explains how to update the properties of the RadioButton field added to the document from the form designer toolbar.

STANDALONE

```
<ejs-pdfviewer
id="pdfViewer"
[documentPath]="document"
[radioButtonFieldSettings] = "radioButtonFieldSettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer
id="pdfViewer"
[serviceUrl]="service"
[documentPath]="document"
[radioButtonFieldSettings] = "radioButtonFieldSettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

```
`typescript
// Properties to customize the RadioButton field settings
public radioButtonFieldSettings = {
// Set the name of the form field element.
name: 'RadioButton',
// Specify whether the RadioButton field is in read-only or read-write mode.
isReadOnly: false,
// Set the visibility of the form field.
visibility: 'visible',
// Specify whether the field is mandatory or not.
isRequired: false,
// Specify whether to print the RadioButton field.
isPrint: true,
// Set the text to be displayed as a tooltip.
tooltip: 'RadioButton',
// Set the thickness of the RadioButton field. To hide the borders, set the value to 0 (zero).
thickness: 4,
// Specifies whether the radio button is in checked state or not.
isSelected: true,
// Set the background color of the radio button in hexadecimal string format.
backgroundColor: 'white',
// Set the border color of the radio button field.
borderColor: 'black',
// Set the value of the form field element.
value: 'RadioButton'
};
`
```



[ListBox field settings](#)

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the ListBox field properties on a button click.


```

`html
<button (click)="updateProperties()">Update Properties</button>
,

`typescript
// Event triggers on the Update Properties button click.
updateProperties() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
var formField = viewer.retrieveFormFields();
var customOptions = [{itemName:'item1',itemValue:'item1'}, {itemName:'item2',itemValue:'item2'},
{itemName:'item3',itemValue:'item3'}]
viewer.formDesignerModule.updateFormField(formField[0], {
name: 'ListBox',
isReadOnly: true,
visibility: 'visible',
isRequired: false,
isPrint: true,
tooltip: 'ListBox',
thickness: 4,
fontFamily: 'Courier',
fontSize: 10,
fontStyle: 'None',
color: 'black',
borderColor: 'black',
backgroundColor: 'white',
alignment: 'Left',
options: customOptions,
});
}
,

```

The following code example explains how to update the properties of the ListBox field added to the document from the form designer toolbar.

STANDALONE

```

<ejs-pdfviewer
id="pdfViewer"
[documentPath]="document"

```

```
[listBoxFieldSettings] = "listBoxFieldSettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer
id="pdfViewer"
[serviceUrl]="service"
[documentPath]="document"
[listBoxFieldSettings] = "listBoxFieldSettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

`typescript

// Properties to customize the ListBox field settings

```
public listBoxFieldSettings = {
```

```
var customOptions = [{itemName:'item1',itemValue:'item1'}, {itemName:'item2',itemValue:'item2'},
{itemName:'item3',itemValue:'item3'}]
```

// Set the name of the form field element.

```
name: 'ListBox',
```

// Specify whether the ListBox field is in read-only or read-write mode.

```
isReadOnly: false,
```

// Set the visibility of the form field.

```
visibility: 'visible',
```

// Specify whether the field is mandatory or not.

```
isRequired: false,
```

// Specify whether to print the ListBox field.

```
isPrint: true,
```

// Set the text to be displayed as a tooltip.

```
tooltip: 'ListBox',
```

// Set the thickness of the ListBox field. To hide the borders, set the value to 0 (zero).

```
thickness: 4,
```

// Set the value of the form field element.

```
value:'ListBox',
```

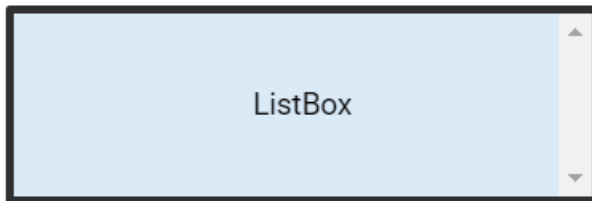
// Set the font family of the ListBox field.

```
fontFamily: 'Courier',
```

// Set the font size of the ListBox field.

```
fontSize: 10,
```

```
// Specify the font style
fontStyle: 'None',
// Set the font color of the ListBox field.
color: 'black',
// Set the border color of the ListBox field.
borderColor: 'black',
// Set the background color of the ListBox field.
backgroundColor: 'White',
// Set the alignment of the text.
alignment: 'Left',
// Set the listbox items.
options: customOptions
};
`
```



[DropDown field settings](#)

Using the `updateFormField` method, the form fields can be updated programmatically.

The following code example explains how to update the DropDown field properties on a button click.

```
`html
<button (click)="updateProperties()">Update Properties</button>
`

`typescript
// Event triggers on the Update Properties button click.
updateProperties() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
var formField = viewer.retrieveFormFields();
var customOptions = [{itemName:'item1',itemValue:'item1'}, {itemName:'item2',itemValue:'item2'},
{itemName:'item3',itemValue:'item3'}]
viewer.formDesignerModule.updateFormField(formField[0], {
```

```

name: 'DropDown',
isReadOnly: true,
visibility: 'visible',
isRequired: false,
isPrint: true,
tooltip: 'DropDown',
thickness: 4,
fontFamily: 'Courier',
fontSize: 10,
fontStyle: 'None',
color: 'black',
borderColor: 'black',
backgroundColor: 'white',
alignment: 'Left',
options: customOptions,
});
}
`

```

The following code example explains how to update the properties of the DropDown field added to the document from the form designer toolbar.

STANDALONE

```

<ejs-pdfviewer
  id="pdfViewer"
  [documentPath]="document"
  [DropDownFieldSettings] = "DropDownFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>

```

SERVER-BACKED

```

<ejs-pdfviewer
  id="pdfViewer"
  [serviceUrl]="service"
  [documentPath]="document"
  [DropDownFieldSettings] = "DropDownFieldSettings"
  style="height:640px;display:block">
</ejs-pdfviewer>

```

`typescript

// Properties to customize the DropDown field settings

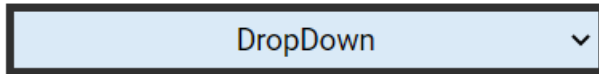
```
public DropdownFieldSettings = {  
var customOptions = [{itemName:'item1',itemValue:'item1'}, {itemName:'item2',itemValue:'item2'},  
{itemName:'item3',itemValue:'item3'}]  
// Set the name of the form field element.  
name: 'DropDown',  
// Specify whether the DropDown field is in read-only or read-write mode.  
isReadOnly: false,  
// Set the visibility of the form field.  
visibility: 'visible',  
// Specify whether the field is mandatory or not.  
isRequired: false,  
// Specify whether to print the DropDown field.  
isPrint: true,  
// Set the text to be displayed as a tooltip.  
tooltip: 'DropDown',  
// Set the thickness of the DropDown field. To hide the borders, set the value to 0 (zero).  
thickness: 4,  
// Set the value of the form field element.  
value:'DropDown',  
// Set the font family of the DropDown field.  
fontFamily: 'Courier',  
// Set the font size of the DropDown field.  
fontSize: 10,  
// Specify the font style  
fontStyle: 'None',  
// Set the font color of the DropDown field.  
color: 'black',  
// Set the border color of the DropDown field.  
borderColor: 'black',  
// Set the background color of the DropDown field.  
backgroundColor: 'White',  
// Set the alignment of the text.  
alignment: 'Left',
```

```
// Set the DropDown items.
```

```
options: customOptions
```

```
};
```

```
,
```



Create form fields with UI interaction

The PDF viewer control provides the option for interaction with Form Fields such as Drag and resize. you can draw a Form Field dynamically by clicking the Form Field icon on the toolbar and draw it in the PDF document. The Form Fields type supported by the PDF Viewer Control are:

- Textbox
- Password
- CheckBox
- RadioButton
- ListBox
- DropDown
- SignatureField
- InitialField

Enable or Disable form designer toolbar

We should inject FormDesigner module and set enableFormDesignerToolbar as true to enable the Form designer icon on the toolbar. By default, enableFormDesignerToolbar is set as true. Use the following code to inject FormDesigner module and to enable the enableFormDesignerToolbar property.

STANDALONE

```
import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService,
FormFieldsService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enableFormDesignerToolbar]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
```

```

TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
  ngOnInit(): void {
  }
}

```

SERVER-BACKED

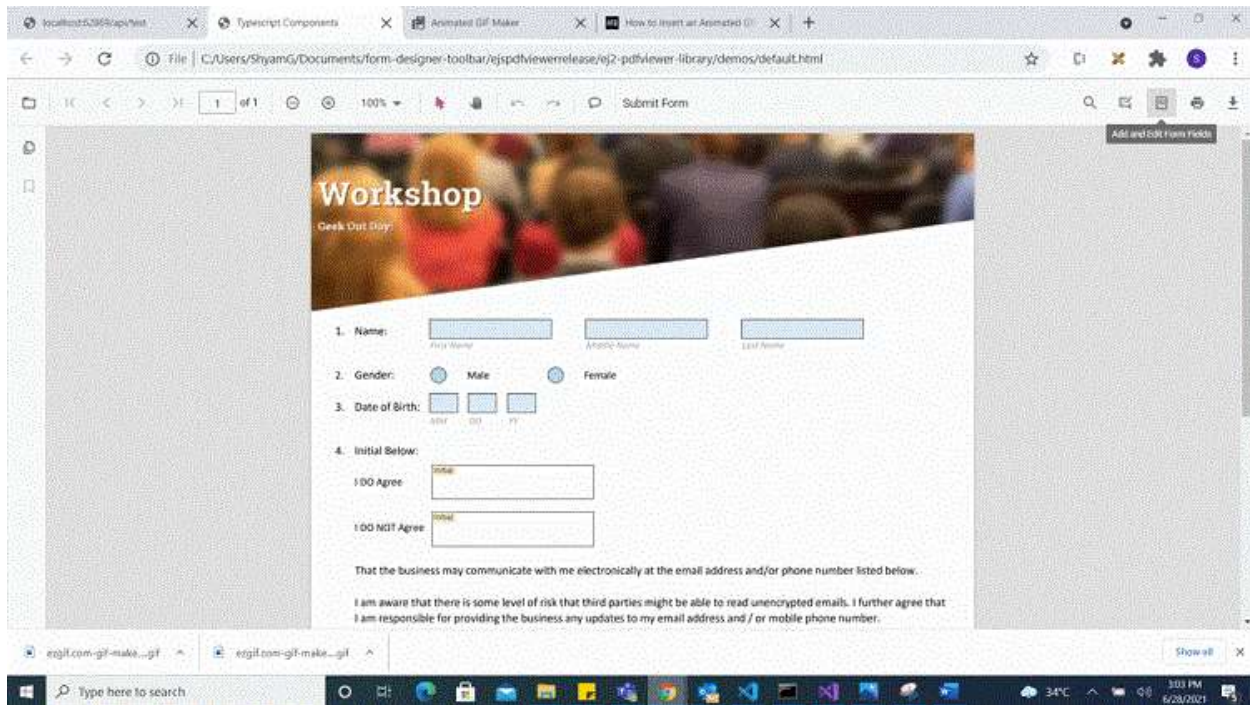
```

import { ViewChild } from '@angular/core';
import { Component, OnInit } from '@angular/core';
import { PdfViewerComponent, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService, ToolbarService,
NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService,
FormFieldsService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // Specifies the template string for the PDF Viewer component.
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enableFormDesignerToolbar]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  @ViewChild('pdfviewer')
  public pdfviewerControl: PdfViewerComponent;
  public service: string =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
  ngOnInit(): void {
  }
}

```

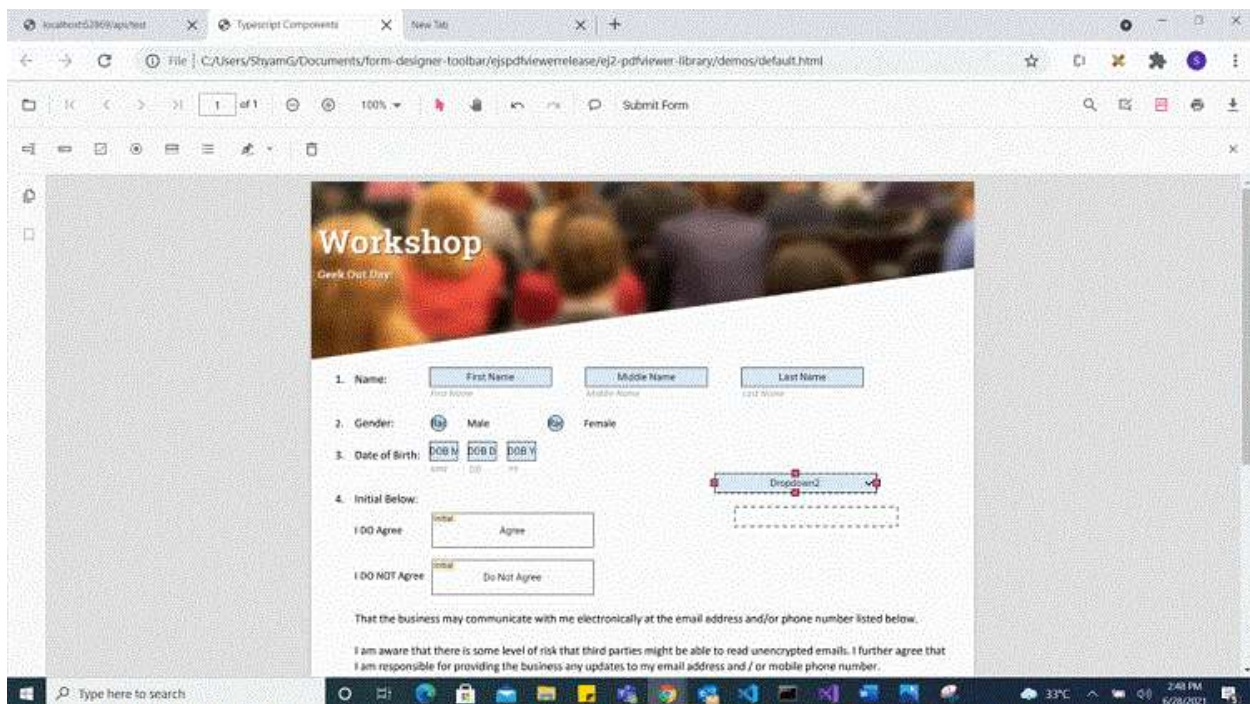
Add the form field dynamically

Click the Form Field icon on the toolbar and then click on to the PDF document to draw a Form Field. Refer the below GIF for further reference.



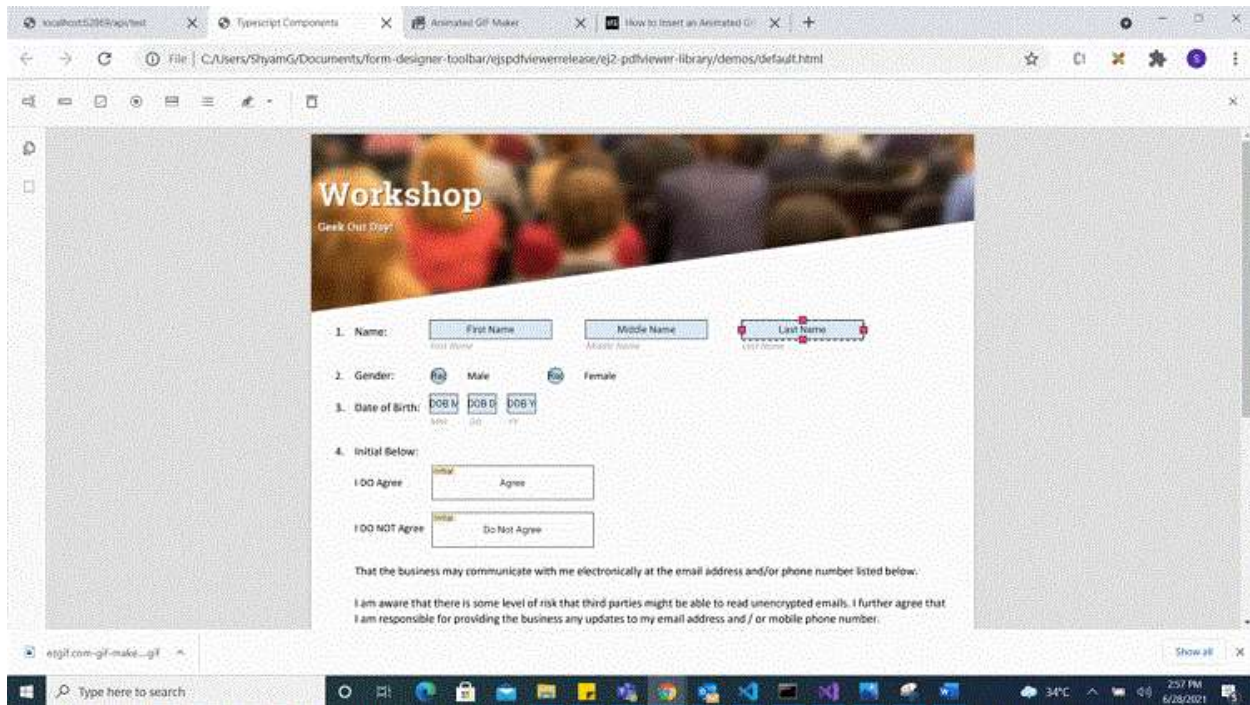
Drag the form field

We provide options to drag the Form Field which is currently selected in the PDF document. Refer the below GIF for further reference.



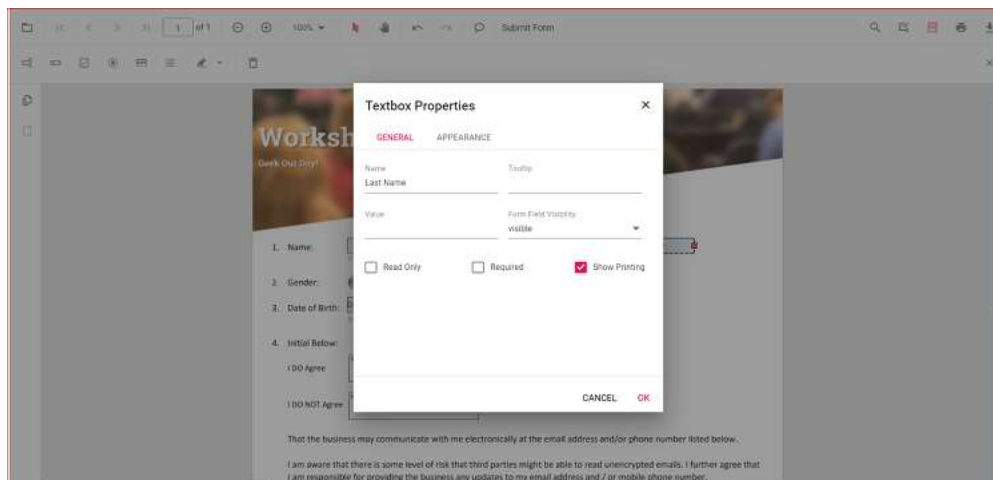
Resize the form field

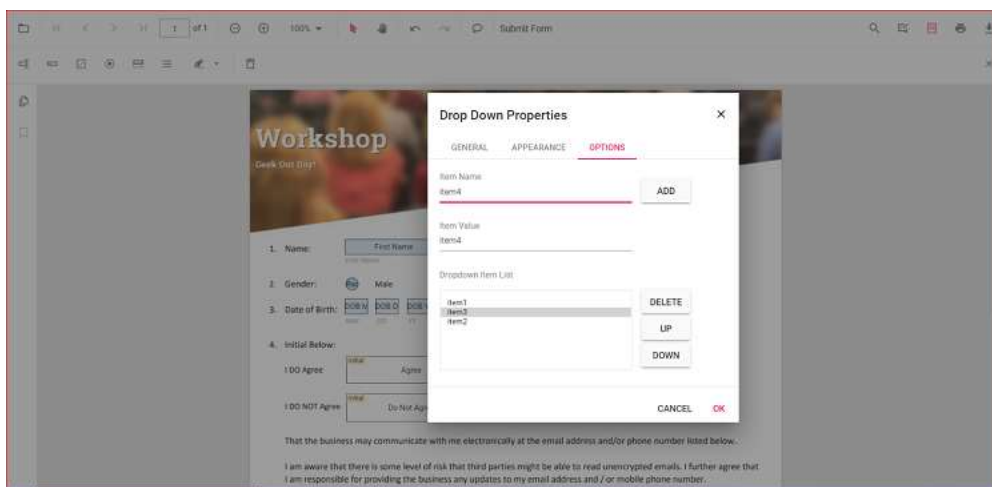
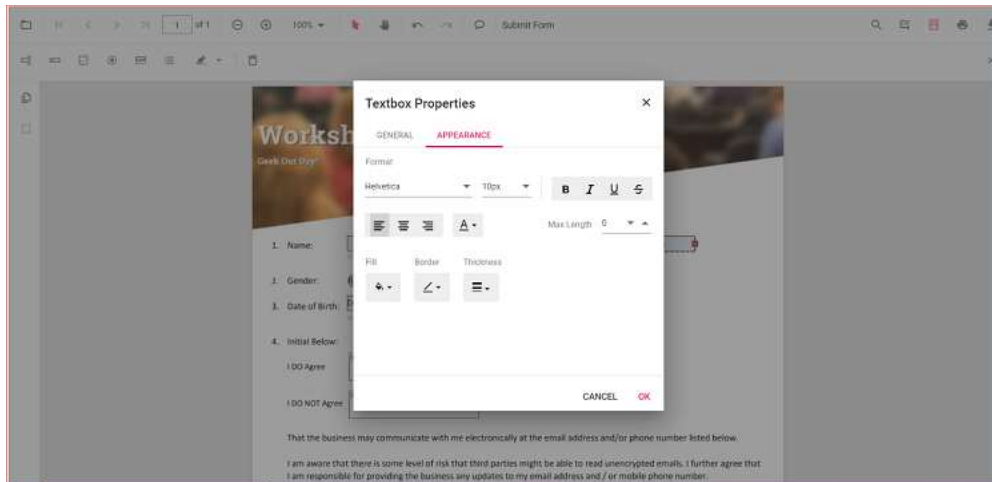
We provide options to resize the Form Field which is currently selected in the PDF document. Refer the below GIF for further reference.



Edit or Update the form field dynamically

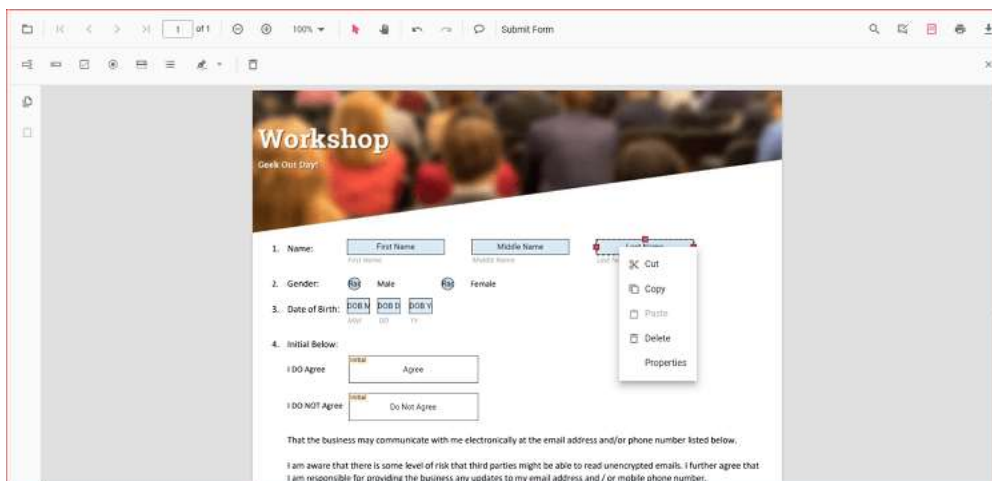
The properties of the Form Fields can be edited using the Form Field Properties window. It can be opened by selecting the Properties option in the context menu that appears on the right by clicking the Form Field object. Refer the below image for the properties available to customize the appearance of the Form Field.





Clipboard operation with form field

The PDF Viewer control supports the clipboard operations such as cut, copy and paste for Form Fields. You can right click on the Form Field object to view the context menu and select to the clipboard options that you would like to perform. Refer the below image for the options in the context menu.



Undo and Redo

We provided support to undo/redo the Form Field actions that are performed at runtime. Use the following code example to perform undo/redo actions.

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.undo();
pdfViewer.redo();
}
</script>
`
```

Print in Angular PDF Viewer component

The PDF Viewer supports printing the loaded PDF file. You can enable/disable the print using the following code snippet.

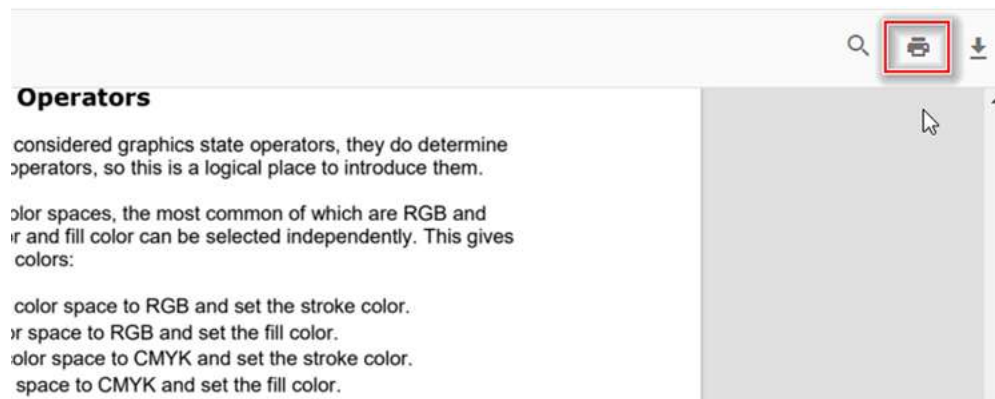
STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[enablePrint]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
```

```
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[enablePrint]='true'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```



You can invoke print action using the following code snippet.,

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.print.print();
}
</script>
```

Print the PDF document in the new window.

PDF Viewer extension supports printing functionality for loaded PDF files directly within the browser.

You can utilize the `printMode` parameter to specify the printing mode, with the option to choose `NewWindow` for printing. Below is a code snippet demonstrating how to implement this functionality

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[printMode] = "printMode"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
  public printMode: string = "NewWindow";
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[printMode] = "printMode"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
  public printMode: string = "NewWindow";
}
```

```

</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public printMode: string = "NewWindow";
}

```

By setting the printMode to **NewWindow**, the extension will open a new window for printing the PDF document, providing a seamless and user-friendly printing experience.

Limiting the Dialog Opening for Printing

In the Syncfusion PDF Viewer, you can control the printing process by leveraging the **printStart** event. This event enables you to customize the printing behavior, particularly restricting the dialog opening. Below is a code snippet demonstrating how to utilize this event

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
(printStart)="printStart($event)"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
ngOnInit(): void {
}
printStart(args: any) {

```

```
console.log(args);
args.cancel = true;
}
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
(printStart)="printStart($event)"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  ngOnInit(): void {
  }
  printStart(args: any) {
    console.log(args);
    args.cancel = true;
  }
}
```

In this code snippet, the `printStart` function is defined to handle the `printStart` event. By setting `args.cancel` to `true`, the print dialog opening is restricted. By default, the `cancel` property is set to `false`.

See also

- [Toolbar items](#)
- [Feature Modules](#)

Download in Angular PDF Viewer component

The PDF Viewer supports downloading the loaded PDF file. You can enable/disable the download using the following code snippet.

STANDALONE

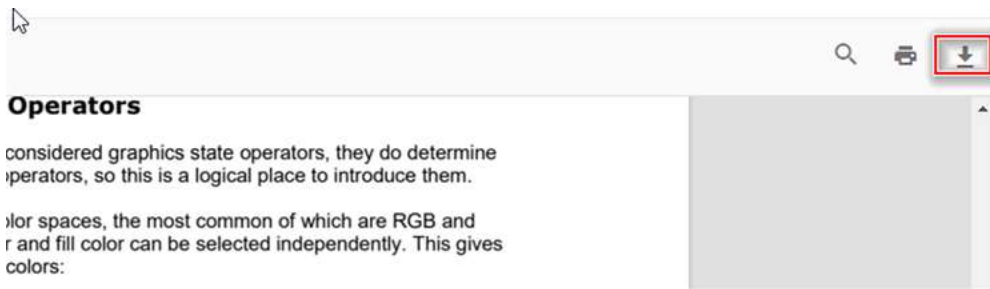
```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
enableDownload='true'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
enableDownload='true'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
```



```
PrintService]
})
export class AppComponent implements OnInit {
  public service =
    'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
    succinctly.pdf';
}
```



You can invoke download action using following code snippet.,

```
`html
<script>
window.onload = function () {
var pdfViewer = document.getElementById('pdfviewer').ej2_instances[0];
pdfViewer.download();
}
</script>
`
```

See also

- [Toolbar items](#)
- [Feature Modules](#)

Localization in Angular PDF Viewer component

The text contents provided in the PDF Viewer can be localized using the collection of localized strings for different cultures. By default, the PDF Viewer is localized in “**en-US**”.

The following table shows the default text values used in PDF Viewer in 'en-US' culture:

Keywords	Values
---	---
PdfViewer	PDF Viewer
Cancel	Cancel
Download file	Download file
Download	Download

Enter Password	This document is password protected. Please enter a password.
File Corrupted	File corrupted
File Corrupted Content	The file is corrupted and cannot be opened.
Fit Page	Fit page
Fit Width	Fit width
Automatic	Automatic
Go To First Page	Show first page
Invalid Password	Incorrect password. Please try again.
Next Page	Show next page
OK	OK
Open	Open file
Page Number	Current page number
Previous Page	Show previous page
Go To Last Page	Show last page
Zoom	Zoom
Zoom In	Zoom in
Zoom Out	Zoom out
Page Thumbnails	Page thumbnails
Bookmarks	Bookmarks
Print	Print file
Password Protected	Password required
Copy	Copy
Text Selection	Text selection tool
Panning	Pan mode
Text Search	Find text
Find in document	Find in document
Match case	Match case
Apply	Apply
GoToPage	Go to page
No matches	Viewer has finished searching the document. No more matches were found
No Text Found	No Text Found
Undo	Undo
Redo	Redo

Annotation	Add or Edit annotations
Highlight	Highlight Text
Underline	Underline Text
Strikethrough	Strikethrough Text
Delete	Delete annotation
Opacity	Opacity
Color edit	Change Color
Opacity edit	Change Opacity
Highlight context	Highlight
Underline context	Underline
Strikethrough context	Strike through
Server error	Web-service is not listening. PDF Viewer depends on web-service for all it's features.
Please start the web service to continue.	
Open text	Open
First text	First Page
Previous text	Previous Page
Next text	Next Page
Last text	Last Page
Zoom in text	Zoom In
Zoom out text	Zoom Out
Selection text	Selection
Pan text	Pan
Print text	Print
Search text	Search
Annotation Edit text	Edit Annotation

The different locale value for the PDF Viewer can be specified using the locale property.

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"

```

```

locale='ar-AE'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
selector: 'app-container',
// specifies the template string for the PDF Viewer component
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
locale='ar-AE'
[documentPath]='document'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
}

```

You have to map the text content based on locale like following script in sample level.,

```
`html
```

```
<script>
```

```
ej.base.L10n.load({
```

```
'ar-AE': {  
  'PdfViewer': {  
    'PdfViewer': 'قوات الدفاع الشعبي المشاهد',  
    'Cancel': 'إلغاء',  
    'Download file': 'تحميل الملف',  
    'Download': 'تحميل',  
    'Enter Password': 'هذا المستند محمي بكلمة مرور. يرجى إدخال كلمة مرور',  
    'File Corrupted': 'ملف تالف',  
    'File Corrupted Content': 'الملف تالف ولا يمكن فتحه',  
    'Fit Page': 'لائق بدنيا الصفحة',  
    'Fit Width': 'لائق بدنيا عرض',  
    'Automatic': 'تلقائي',  
    'Go To First Page': 'عرض الصفحة الأولى',  
    'Invalid Password': 'كلمة سر خاطئة. حاول مرة أخرى',  
    'Next Page': 'عرض الصفحة التالية',  
    'OK': 'حسنًا',  
    'Open': 'فتح الملف',  
    'Page Number': 'رقم الصفحة الحالية',  
    'Previous Page': 'عرض الصفحة السابقة',  
    'Go To Last Page': 'عرض الصفحة الأخيرة',  
    'Zoom': 'تكبير',  
    'Zoom In': 'تكبير في',  
    'Zoom Out': 'تكبير خارج',  
    'Page Thumbnails': 'مصغرات الصفحة',  
    'Bookmarks': 'المرجعية',  
    'Print': 'اطبع الملف',  
    'Password Protected': 'كلمة المرور مطلوبة',  
    'Copy': 'نسخ',  
    'Text Selection': 'أداة اختيار النص',  
    'Panning': 'وضع عموم',  
    'Text Search': 'بحث عن نص',  
    'Find in document': 'ابحث في المستند',  
    'Match case': 'حالة مباراة',
```

'Apply': 'تطبيق',
'GoToPage': 'انتقل إلى صفحة',
// tslint:disable-next-line:max-line-length
'No matches': 'انتهى العارض من البحث في المستند. لم يتم العثور على مزيد من التطابقات',
'No Text Found': 'لم يتم العثور على نص',
'Undo': 'فك',
'Redo': 'فعل ثانية',
'Annotation': 'إضافة أو تعديل التعليقات التوضيحية',
'Highlight': 'تسليط الضوء على النص',
'Underline': 'تسطير النص',
'Strikethrough': 'نص يتوسطه خط',
'Delete': 'حذف التعليق التوضيحي',
'Opacity': 'غموض',
'Color edit': 'غير اللون',
'Opacity edit': 'تغيير التعقيم',
'Highlight context': 'تسليط الضوء',
'Underline context': 'أكد',
'Strikethrough context': 'يتوسطه',
// tslint:disable-next-line:max-line-length
'Server error': 'خدمة الانترنت لا يستمع. يعتمد قوات الدفاع الشعبي المشاهد على خدمة الويب لجميع ميزاته. يرجى بدء خدمة'.
'Open text': 'افتح',
'First text': 'الصفحة الأولى',
'Previous text': 'الصفحة السابقة',
'Next text': 'الصفحة التالية',
'Last text': 'آخر صفحة',
'Zoom in text': 'تكبير',
'Zoom out text': 'تصغير',
'Selection text': 'اختيار',
'Pan text': 'مقالة',
'Print text': 'طباعة',
'Search text': 'بحث',
'Annotation Edit text': 'تحرير التعليق التوضيحي'

```

}
}
});
</script>
`

```

Accessibility in Syncfusion Angular PDF Viewer components

The PDF Viewer 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 PDF Viewer component is outlined below.

```

| Accessibility Criteria | Compatibility |
| -- | -- |
| WCAG 2.2 Support |  |
| Section 508 Support |  |
| Screen Reader Support |  |
| Right-To-Left Support |  |
| Color Contrast |  |
| Mobile Device Support |  |
| Keyboard Navigation Support |  |
| Accessibility Checker Validation |  |
| Axe-core Accessibility Validation |  |
<style>
.post .post-content img {
display: inline-block;
margin: 0.5em 0;
}
</style>
<div> - All
features of the component meet the requirement.</div>

```

```
<div> - Some features of the component do not meet the requirement.</div>
```

```
<div> - The component does not meet the requirement.</div>
```

WAI-ARIA attributes

[WAI-ARIA](#) (Accessibility Initiative – Accessible Rich Internet Applications) defines a way to increase the accessibility of web pages, dynamic content, and user interface components developed with Ajax, HTML, JavaScript, and related technologies. ARIA provides additional semantics to describe the role, state, and functionality of web components. The following ARIA attributes are used in the PDF Viewer component:

Attributes	Purpose
---	---
<code>aria-disabled</code>	Indicates whether the PDF Viewer component is in a disabled state or not.
<code>aria-expanded</code>	Indicates whether the suggestion list has expanded or not.
<code>aria-readonly</code>	Indicates the readonly state of the PDF Viewer element.
<code>aria-haspopup</code>	Indicates whether the PDF Viewer input element has a suggestion list or not.
<code>aria-label</code>	Indicates the breadcrumb item text.
<code>aria-labelledby</code>	Provides a label for the PDF Viewer. Typically, the "aria-labelledby" attribute will contain the id of the element used as the PDF Viewer's title.
<code>aria-describedby</code>	This attribute points to the PDF Viewer element describing the one it's set on.
<code>aria-orientation</code>	Indicates whether the PDF Viewer element is oriented horizontally or vertically.
<code>aria-valuetext</code>	Returns the current text of the PDF Viewer.
<code>aria-valuemax</code>	Indicates the Maximum value of the PDF Viewer.
<code>aria-valuemin</code>	Indicates the Minimum value of the PDF Viewer.
<code>aria-valuenow</code>	Indicates the current value of the PDF Viewer.
<code>aria-controls</code>	Attribute is set to the button and it points to the corresponding content.

Keyboard interaction

The PDF Viewer component followed the [keyboard interaction](#) guideline, making it easy for people who use assistive technologies (AT) and those who completely rely on keyboard navigation. The following keyboard shortcuts are supported by the Message component.

Press (Windows)	Press (Macintosh)	To do this
-----------------	-------------------	------------

---	---	---
-----	-----	-----

|| **Shortcuts for page navigation** |

CONTROL + Left Arrow (or) CONTROL + Up Arrow	COMMAND + Left Arrow (or) COMMAND + Up Arrow	Navigate to the first page
----------------------------------------------	----------------------------------------------	----------------------------

CONTROL + Right Arrow (or) CONTROL + Down Arrow	COMMAND + Right Arrow (or) COMMAND + Down Arrow	Navigate to the last page
-------------------------------------------------	-------------------------------------------------	---------------------------

| Left Arrow | Left Arrow (or) Shift + Space | Navigate to the previous page |

| Right Arrow | Right Arrow (or) Space | Navigate to the next page |

| CONTROL + G | COMMAND + G | Go To The Page |

| Up Arrow | Up Arrow | Scroll up |

| Down Arrow | Down Arrow | Scroll down |

|| Shortcuts for Zooming |

| CONTROL + = | COMMAND + = | Perform zoom-in operation |

| CONTROL + - | COMMAND + - | Perform zoom-out operation |

| CONTROL + 0 | COMMAND + 0 | Retain the zoom level to 1 |

|| Shortcut for Text Search |

| CONTROL + F | COMMAND + F | Open the search toolbar |

|| Shortcut for Text Selection |

| CONTROL + C | CONTROL + C | Copy the selected text or annotation or form field |

| CONTROL + X | CONTROL + X | Cut the selected text or annotation of the form field |

| CONTROL + Y | CONTROL + Y | Paste the selected text or annotation or form field |

|| Shortcuts for the general operation |

| CONTROL + Z | CONTROL + Z | Undo the action |

| CONTROL + Y | CONTROL + Y | Redo the action |

| CONTROL + P | COMMAND + P | Print the document |

| Delete | Delete | Delete the annotations and form fields |

| CONTROL + Shift + A | COMMAND + Shift + A | Toggle Annotation Toolbar |

| CONTROL + Alt + 0 | COMMAND + Option + 0 | Show Command panel |

| CONTROL + Alt + 2 | COMMAND + Option + 2 | Show Bookmarks |

| CONTROL + Alt + 1 | COMMAND + Option + 1 | Show Thumbnails |

| CONTROL + S | COMMAND + S | Download |

| Shift + H | Shift + H | Enable pan mode |

| Shift + V | Shift + V | Enable text selection mode |

The current implementation of our PDF Viewer includes keyboard shortcuts for various functions like scrolling, zooming, text search, printing, and annotation deletion.

To enhance user experience, we're adding additional keyboard shortcuts for actions such as navigating between pages, accessing specific pages, toggling annotation tools, and displaying PDF elements like outlines, annotations, bookmarks, and thumbnails.

To support this, we're introducing a new class called **commandManager**, which handles custom commands triggered by specific key gestures. These custom commands will be defined by users and executed accordingly.

The **commandManager** will have a parameter called **Commands**, which will hold the collection of custom keyboard commands specified by users. Each custom command will be represented by a **KeyboardCommand** class, containing the **command name** and associated **keyboard combination**.

Additionally, we're introducing the **keyboardCustomCommands** parameter for the **CommandManager**, which will utilize the **EventCallback** to handle keyboard events and trigger appropriate methods when specific key combinations are pressed.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resourceUrl'
[commandManager]='commandManager'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resourceUrl = 'https://cdn.syncfusion.com/ej2/24.1.41/dist/ej2-
pdfviewer-lib';
  ngOnInit(): void {
  }
  public commandManager = {
    keyboardCommand: [{
      name: 'customCopy',
      gesture: {
        pdfKeys: PdfKeys.G,
        modifierKeys: ModifierKeys.Shift | ModifierKeys.Alt
      }
    },
    {
      name: 'customPaste',
      gesture: {
        pdfKeys: PdfKeys.H,
        modifierKeys: ModifierKeys.Shift | ModifierKeys.Alt
      }
    }
  ]
}
```

```

    },
    {
      name: 'customCut',
      gesture: {
        pdfKeys: PdfKeys.Z,
        modifierKeys: ModifierKeys.Control
      }
    },
    {
      name: 'customSelectAll',
      gesture: {
        pdfKeys: PdfKeys.E,
        modifierKeys: ModifierKeys.Control
      }
    },
  ],
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, AnnotationService, TextSelectionService,
PrintService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-container',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[commandManager]='commandManager'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  ngOnInit(): void {
  }
  public commandManager = {
    keyboardCommand: [{
      name: 'customCopy',
      gesture: {

```

```
pdfKeys: PdfKeys.G,  
modifierKeys: ModifierKeys.Shift | ModifierKeys.Alt  
},  
{  
  name: 'customPaste',  
  gesture: {  
    pdfKeys: PdfKeys.H,  
    modifierKeys: ModifierKeys.Shift | ModifierKeys.Alt  
  },  
{  
  name: 'customCut',  
  gesture: {  
    pdfKeys: PdfKeys.Z,  
    modifierKeys: ModifierKeys.Control  
  },  
{  
  name: 'customSelectAll',  
  gesture: {  
    pdfKeys: PdfKeys.E,  
    modifierKeys: ModifierKeys.Control  
  },  
],  
}
```

Each `keyboardCommand` object consists of a `name` property, specifying the name of the custom command, and a `gesture` property, defining the key gesture associated with the command.

For example, the first command named `customCopy` is associated with the **G** key and requires both the **Shift** and **Alt** modifier keys to be pressed simultaneously.

Additionally, there's an explanation of the key modifiers used in the gestures:

- Ctrl corresponds to the Control key, represented by the value **1**.
- Alt corresponds to the Alt key, represented by the value **2**.
- Shift corresponds to the Shift key, represented by the value **4**.
- Meta corresponds to the Command key on macOS or the Windows key on Windows, represented by the value **8**.

This setup allows users to perform custom actions within the PDF viewer by pressing specific key combinations, enhancing the user experience and providing more efficient navigation and interaction options.

Ensuring accessibility

The PDF Viewer component's accessibility levels are ensured through an [accessibility-checker](#) and [axe-core](#) software tools during automated testing.

The accessibility compliance of the PDF Viewer component is shown in the following sample. Open the [sample](#) in a new window to evaluate the accessibility of the PDF Viewer component with accessibility tools.

See also

- [Accessibility in Syncfusion Angular components](#)

How To

Customize the toolbar

The PDF Viewer provides API for user interactions options provided in its built-in toolbar. Using this, you can create your own User Interface for toolbar actions at the application level by hiding the default toolbar. The following steps are used to create the custom toolbar for PDF Viewer:

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: Hide the default toolbar of PDF Viewer using the following code snippet.,

STANDALONE

```
<ejs-pdfviewer #pdfviewer id='pdfViewer'
[documentPath]='document'
[enableToolbar]=false
[enableNavigationToolbar]=false
(pageChange)='pageChanged($event)'
(documentLoad)='documentLoaded($event)'
style="height:640px; display: block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer #pdfviewer id='pdfViewer'
[serviceUrl]='service'
[documentPath]='document'
[enableToolbar]=false
[enableNavigationToolbar]=false
(pageChange)='pageChanged($event)'
(documentLoad)='documentLoaded($event)'
style="height:640px; display: block">
</ejs-pdfviewer>
```

Step 3: Add EJ2 toolbar for performing primary actions like Open, Previous page, Next page, Go to page, Print and Download using the following code snippet.

`html

```
<ejs-toolbar id='topToolbar' #customToolbar>
<e-items>
<e-item prefixIcon='e-pv-open-document-icon' id='openDocument' tooltipText='Open'
(click)='openDocument($event)'>
</e-item>
```

```

<e-item prefixIcon='e-pv-previous-page-navigation-icon' id='previousPage' tooltipText='Previous Page'
(click)='previousClicked($event)'
align='Center'></e-item>

<e-item prefixIcon='e-pv-next-page-navigation-icon' id='nextPage' tooltipText='Next Page'
(click)='nextClicked($event)' align='Center'></e-item>

<e-item align='Center'>
<ng-template #template>
<div class=''>
<input type='text' class='e-input-group e-pv-current-page-number' id='currentPage' value='0'
(keypress)='onCurrentPageBoxKeypress($event)' (click)='onCurrentPageBoxClicked($event)' />
</div>
<div style='margin-left: 6px'><span class='e-pv-total-page-number' id='totalPage'>of 0</span></div>
</ng-template>
</e-item>

<e-item prefixIcon='e-pv-print-document-icon' tooltipText='Print' (click)='printClicked($event)'
align='Right'></e-item>

<e-item prefixIcon='e-pv-download-document-icon' tooltipText='Download'
(click)='downloadClicked($event)' align='Right'></e-item>

</e-items>
</ejs-toolbar>

<input type="file" id="fileUpload" accept=".pdf"
style="display:block;visibility:hidden;width:0;height:0;">
,

```

Step 3: Add EJ2 toolbar for performing magnification actions in PDF Viewer using the following code snippet.

```

`html
<ejs-toolbar id='magnificationToolbar' #zoomToolbar>
<e-items>

<e-item prefixIcon='e-pv-fit-page-icon' id='fitPage' tooltipText='Fit to page'
(click)='pageFitClicked($event)'></e-item>

<e-item prefixIcon='e-pv-zoom-in-icon' id='zoomIn' tooltipText='Zoom in'
(click)='zoomInClicked($event)'></e-item>

<e-item prefixIcon='e-pv-zoom-out-sample' id='zoomOut' tooltipText='Zoom out'
(click)='zoomOutClicked($event)'></e-item>

</e-items>
</ejs-toolbar>

```

Step 4: Add the following style to achieve the custom toolbar styling.

```
`css
magnificationToolbar {
background: transparent;

height: auto;
min-height: 56px;
width: 200px;
border: none;
position: absolute;
z-index: 1001;
bottom: 58px;
right: 16px;
transform: rotate(90deg);
}

magnificationToolbar.e-toolbar .e-toolbar-items {
background: transparent;
}

magnificationToolbar.e-toolbar .e-tbar-btn {
border-radius: 50%;
min-height: 30px;
min-width: 30px;
border: 1px solid #c8c8c8;
}

topToolbar {
top: 0px;
z-index: 1001;
}

.e-tbar-section .e-sample-resize-container {
height: 46px;
}

.e-bookmark-popup {
height: 200px;
max-width: 250px;
```

```
}  
.e-text-search-popup {  
height: 104px;  
max-width: 348px;  
}  
.e-custom-search-input {  
width: 234px;  
}  
.e-text-search-popup .e-footer-content, .e-bookmark-popup .e-footer-content {  
padding: 0;  
height: 0;  
}  
.search-button, .search-button:disabled, .search-button:focus, .search-button:hover {  
background: transparent;  
box-shadow: none;  
}  
popup .e-dlg-content {  
padding-left: 0;  
padding-bottom: 0;  
}  
.e-pv-bookmarks {  
min-width: 234px;  
}  
.e-pv-current-page-number {  
width: 46px;  
height: 28px;  
text-align: center;  
}  
.material .e-pv-current-page-number {  
border-width: 1px;  
}  
.e-icons {  
font-family: "e-icons";
```



```
font-style: normal;
font-variant: normal;
font-weight: normal;
line-height: 1;
text-transform: none;
}
.e-pv-icon::before {
font-family: 'e-icons';
}
.e-pv-icon-search::before {
font-family: 'e-icons';
font-size: 12px;
}
.e-pv-download-document-icon::before {
content: '\e914';
}
.e-pv-print-document-icon::before {
content: '\e917';
}
.e-pv-previous-page-navigation-icon::before {
content: '\e910';
}
.e-pv-next-page-navigation-icon::before {
content: '\e911';
}
.e-pv-zoom-out-sample::before {
content: '\e912';
}
.e-pv-zoom-out-sample {
transform: rotate(90deg);
padding-right: 2px;
}
.e-pv-zoom-in-icon::before {
```

```

content: '\e91d';
}
.e-pv-fit-page-icon::before {
content: '\e91b';
}
.e-pv-open-document-icon::before {
content: '\e91c';
}
@font-face {
font-family: "e-icons";
font-style: normal;
font-weight: normal;
src: url(data:application/x-font-ttf;charset=utf-
8;base64,AAEAAAAKAIAAAwAgT1MvMj8wS0QAAAEoAAAAVmNtYXDSenNLMAAABuAAAAFZnbHlmok0Nt
wAAAjAAAAAPkaGVhZBN3pEcAADQAAAAANmhoZWEHrwNhAAAArAAAACRobXR4NsgAAAAAAAYAAAAA4b
G9jYQdkBmQAAAIQAAAAHm1heHABHAAwAAABCAAAACBuYW1lD0oZXgAABhQAAALBcG9zdFG4mE4AA
AjYAAAAyAABAAADUv9qAFoEAAAA/+gEAAABAAAAAAAAAAAAAAAAAAAAADgABAAAAAQAAxsly1F8PPPU
ACwPoAAAAANgsr7EAAAAA2CyvsQAAAAEAAQAAAAACAACAAAAAAAAAAAAEAAAAOACQABAAAAAAAAG
AAAAAoCgAAAP8AAAAAAAAAAQPqAZAABQAAAanoCvAAAAIwCegK8AAAB4AAxAQIAAAIABQMAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAUGZFZABA6RDpHQNS/2oAWgQAAJYAAAABAAAAAAAAABAAAAAPoAAD6
AAAA+gAAAPoAAD6AAAA+gAAAPoAAD6AAAA+gAAAPoAAD6AAAA+gAAAPoAAAAAACAAAAAwAA
ABQAAwABAAAAFAAEAEIAAAAGAAQAAQAC6RLpHf//AADpEOkU//8AAAAAAAAEABgAKAAAAAQACAAMA
BQAGAAcACAAJAAoACwAMAA0ABAAAAAAAAAAUACoAZACkAL4A7gEuAVwBcAGEAZ4ByAHyAAAAAQAA
AAD6gMuAAUAAAkBBwkBJwIAAet0/on+iXQDL/4VcwF3/olzAAEAAAAAAAA+oDLgAFAAATCQEXCQGJAXcB
d3T+FF4VAy/+iQF3c/4VAesAAAAAAwAAAAEAAQAAAMADwAbAAABITUhBQ4BBY4BJz4BNx4BBRYAFzYA
NyYAJwYAAQACAP4AAoAE2aOj2QQE2aOj2fyEBgEh2dkBIQYG/t/22f7fAcCAQKPZBATZo6PZBATZo9n+3wY
GASHZ2QEhBgb+3wAAAAADAAAAAQABAAACwAXACMAAAEjFTMVMzUzNSM1lwEOAQcuASc+ATceAQ
UWABc2ADcmACcGAHAwMCAwMCAACAE2aOj2QQE2aOj2fyEBgEh2dkBIQYG/t/22f7fAkCAwMCAwP8A
o9kEBNmjo9kEBNmj2f7fBgYBIdnZASEGBv7fAAIAAAAAAwAEAAADAAoAADEhNSEBIQkBIREhAwD9AAEA/
wABgAGA/wD/AIACAP6AAYABgAACAAAAAANABAAADgAaAAABMh4CFREIBRE0Nz4BMycGFREIBRE0JiMh
lgKdCwwHBf7g/uAJBAwKdC8BoAGgX0T+BkQDgAYGCwr9YHZ2AqAOCQQGUS9D/KGrqwNfRIsAAAAACAA
AAP/BAAACwAJAABDgEHLgEnPgE3HgEFHgEXMjY/ARcVATcBlyc3PgE1LgEnDgECgAOQbW2QAwoQbW
2Q/YME2aNGfDIDJAEYf78MyMCKi4E2aOj2QKAbZADA5BtbZADA5Bto9kELioDJDp+/GEBBCQDMnxGo9k
EBNkAAAAQAAAAABAAEAAADAACAFQAZAAABFSE1JRujNSERMxUhnTMRLgEnIQ4BNyE1IQLA/oACQID9A
MACgMABSDf9ADdlvwKA/YABwMDAwICA/sDAwAFAN0gBAUmkwAAAAQAAAAACQAQAAAUAAEBENwk
BJwHsU/6HAXpSamD+YGIBPgE+YgAAAAEAAAAAAkAEAAFAAAARQCQEXCQEBev6HUwHs/hMDnv7C/sJiAa
ABoAABAAAAAAKABAAACwAAERcHFzcXNyc3Jwcn9fVM9PVL9PRL9fQDtfX0TPX1TPT0TPT0AAAAABAAAA
AD8APwAAUACwARABcAAcEzNTM1lQUzFTMRISUhNSM1lwUjFSErIwJ2fvz+hv2K/H7+hgJ2AXr8fv6G/AF6
fvx+fvwBevx+/Px+AXoAAAAAAGAAAAEAAQAAAMAFgAAAREhEScGFREUFhchPgE1ETQmlyEnIQYDgP0AY
h48LQMULTw8Lf5pa/7ULQMA/gACAN8eLf1YLTWdAzwtAigvPYACAAAAAASAN4AAQAAAAAAAABAAA
AAQAAAAAAAQAUAEEAAQAAAAAAAGAHABUAAQAAAAAAAwAUABwAAQAAAAAABAAUADAAAQAAAAA
ABQALAEQAAQAAAAAABgAUAE8AAQAAAAAACgAsAGMAAQAAAAAACwASAI8AAwABBAKAAAAACAKEAA

```

```
wABBAkAAQAoAKMAAwABBAkAAgAOAMsAAwABBAkAAwAoANKAAwABBAkABAAoAQEAawABBAkABQ
AWASKAAwABBAkABgAoAT8AAwABBAkACgBYAWcAAwABBAkACwAkAb8gY3VzdG9tLXRvb2xiYXJbMTkw
OF1SZWd1bGFyY3VzdG9tLXRvb2xiYXJbMTkwOF1jdXN0b20tdG9vbGJhclsxOTA4XVZlcnNpb24gMS4wY3V
zdG9tLXRvb2xiYXJbMTkwOF1Gb250IGdlbmVvYXRIZCB1c2luZyBTeW5jZnVzaW9uIE1ldHJvIFN0dWRpb3d3
dy5zeW5jZnVzaW9uLmNvbQAgAGMAdQBzAHQAbwBtAC0AdABvAG8AbABiAGEAcgBbADEAOQAwADgA
XQBSAGUAZwB1AGwAYQByAGMAdQBzAHQAbwBtAC0AdABvAG8AbABiAGEAcgBbADEAOQAwADgAXQB
jAHUAcwB0AG8AbQAtAHQAbwBvAGwAYgBhAHIAWwAxADkAMAA4AF0AVgBIAHIAcwBpAG8AbgAgADE
ALgAwAGMAdQBzAHQAbwBtAC0AdABvAG8AbABiAGEAcgBbADEAOQAwADgAXQBAG8AbgB0ACAAZw
BIAG4AZQByAGEAdABIAGQAIAB1AHMAaQBuAGcAIABTAHkAbgBjAGYAdQBzAGkAbwBuACAATQBIHQ
cgBvACAAUwB0AHUAZABpAG8AdwB3AHcALgBzAHkAbgBjAGYAdQBzAGkAbwBuAC4AYwBvAG0AAAAA
gAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAOQIBAwEEAQUBBgEHAQgBCQEKAQsBDAE
NAQ4BDwAIVG9wLWlj24LZG93bi1hcnJvdzIKUFZfWm9vbW91dAlQVl9ab29taW4LUFZfRG93bmxyVWQL
UFZfQm9va21hcmsJUFZfU2VhcmNoCFBWX1ByaW50C1BWX1ByZXZpb3VzB1BWX05leHQIUfZfQ2xvc2U
MUFZfRml0VG9QYWdlB1BWX09wZW4AAA==) format('truetype');
```

The icons are embedded in the font file used in the previous code snippet.

Step 5: Add the following code snippet in `app.component.ts` file for performing a user interaction in PDF Viewer in code behind.

STANDALONE

```
@ViewChild('pdfviewer')
public pdfviewerControl: PdfViewerComponent;
@ViewChild('customToolbar')
public customToolbar: ToolbarComponent;
@ViewChild('zoomToolbar')
public zoomToolbar: ToolbarComponent;
public document: string = 'https://cdn.syncfusion.com/content/pdf/hive-
succinctly.pdf';
constructor() { }
ngOnInit(): void {
  // ngOnInit function
  document
    .getElementById('fileUpload')
    .addEventListener('change', this.readFile.bind(this));
}
public openDocument(e: ClickEventArgs): void {
  document.getElementById('fileUpload').click();
}
public previousClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.navigation.goToPreviousPage();
}
public nextClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.navigation.goToNextPage();
}
public printClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.print.print();
}
public downloadClicked(e: ClickEventArgs): void {
  let fileName: string = (document.getElementById(
    'fileUpload'
```

```

) as HTMLInputElement).value
.split('\\')
.pop();
if(fileName !== '') {
this.pdfviewerControl.fileName = fileName;
}
this.pdfviewerControl.download();
}

public pageFitClicked(e: ClickEventArgs): void {
this.pdfviewerControl.magnification.fitToPage();
this.updateZoomButtons();
this.customToolbar.enableItems(document.getElementById('fitPage'), false);
}

public zoomInClicked(e: ClickEventArgs): void {
this.pdfviewerControl.magnification.zoomIn();
this.updateZoomButtons();
}

public zoomOutClicked(e: ClickEventArgs): void {
this.pdfviewerControl.magnification.zoomOut();
this.updateZoomButtons();
}

public pageChanged(e: PageChangeEventArgs): void {
(document.getElementById(
'currentPage'
) as HTMLInputElement).value =
this.pdfviewerControl.currentPageNumber.toString();
this.updatePageNavigation();
}

public documentLoaded(e: LoadEventArgs): void {
document.getElementById('totalPage').textContent =
'of ' + this.pdfviewerControl.pageCount;
(document.getElementById(
'currentPage'
) as HTMLInputElement).value =
this.pdfviewerControl.currentPageNumber.toString();
this.updatePageNavigation();
}

public onCurrentPageBoxClicked(e: ClickEventArgs): void {
(document.getElementById('currentPage') as HTMLInputElement).select();
}

public onCurrentPageBoxKeyPress(e: KeyboardEvent): boolean {
if ((e.which < 48 || e.which > 57) && e.which !== 8 && e.which !== 13) {
e.preventDefault();
return false;
} else {
// tslint:disable-next-line:radix
const currentPageNumber: number = parseInt(
(document.getElementById('currentPage') as HTMLInputElement).value
);
if (e.which === 13) {
if (
currentPageNumber > 0 &&
currentPageNumber <= this.pdfviewerControl.pageCount
) {
this.pdfviewerControl.navigation.goToPage(currentPageNumber);
} else {
// tslint:disable-next-line:max-line-length

```

```

(document.getElementById(
'currentPage'
) as HTMLInputElement).value =
this.pdfviewerControl.currentPageNumber.toString();
}
}
return true;
}
}

private updatePageNavigation(): void {
if(this.pdfviewerControl.currentPageNumber === 1) {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
false
);
this.customToolbar.enableItems(document.getElementById('nextPage'), true);
} else if (
this.pdfviewerControl.currentPageNumber ===
this.pdfviewerControl.pageCount
) {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
true
);
this.customToolbar.enableItems(
document.getElementById('nextPage'),
false
);
} else {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
true
);
this.customToolbar.enableItems(document.getElementById('nextPage'), true);
}
}

private updateZoomButtons(): void {
if(this.pdfviewerControl.zoomPercentage <= 50) {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), true);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), false);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
} else if (this.pdfviewerControl.zoomPercentage >= 400) {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), false);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), true);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
} else {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), true);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), true);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
}
}

// tslint:disable-next-line
private readFile(args: any): void {
// tslint:disable-next-line
let uploadedFiles: any = args.target.files;
if(args.target.files[0] !== null) {
let uploadedFile: File = uploadedFiles[0];

```

```

if (uploadedFile) {
  let reader: FileReader = new FileReader();
  reader.readAsDataURL(uploadedFile);
  // tslint:disable-next-line
  let proxy: any = this;
  // tslint:disable-next-line
  reader.onload = (e: any): void => {
    let uploadedFileUrl: string = e.currentTarget.result;
    proxy.pdfviewerControl.load(uploadedFileUrl, null);
  };
}
}
}

```

SERVER-BACKED

```

@ViewChild('pdfviewer')
public pdfviewerControl: PdfViewerComponent;
@ViewChild('customToolbar')
public customToolbar: ToolbarComponent;
@ViewChild('zoomToolbar')
public zoomToolbar: ToolbarComponent;
public service: string =
  'https://services.syncfusion.com/angular/production/api/pdfviewer';
public document: string = 'https://cdn.syncfusion.com/content/pdf/hive-
succinctly.pdf';
constructor() { }
ngOnInit(): void {
  // ngOnInit function
  document
    .getElementById('fileUpload')
    .addEventListener('change', this.readFile.bind(this));
}
public openDocument(e: ClickEventArgs): void {
  document.getElementById('fileUpload').click();
}
public previousClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.navigation.goToPreviousPage();
}
public nextClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.navigation.goToNextPage();
}
public printClicked(e: ClickEventArgs): void {
  this.pdfviewerControl.print.print();
}
public downloadClicked(e: ClickEventArgs): void {
  let fileName: string = (document.getElementById(
    'fileUpload'
  ) as HTMLInputElement).value
    .split('\\')
    .pop();
  if(fileName !== '') {
    this.pdfviewerControl.fileName = fileName;
  }
  this.pdfviewerControl.download();
}

```

```

public pageFitClicked(e: ClickEventArgs): void {
    this.pdfviewerControl.magnification.fitToPage();
    this.updateZoomButtons();
    this.customToolbar.enableItems(document.getElementById('fitPage'), false);
}
public zoomInClicked(e: ClickEventArgs): void {
    this.pdfviewerControl.magnification.zoomIn();
    this.updateZoomButtons();
}
public zoomOutClicked(e: ClickEventArgs): void {
    this.pdfviewerControl.magnification.zoomOut();
    this.updateZoomButtons();
}
public pageChanged(e: PageChangeEventArgs): void {
    (document.getElementById(
        'currentPage'
    ) as HTMLInputElement).value =
    this.pdfviewerControl.currentPageNumber.toString();
    this.updatePageNavigation();
}
public documentLoaded(e: LoadEventArgs): void {
    document.getElementById('totalPage').textContent =
    'of ' + this.pdfviewerControl.pageCount;
    (document.getElementById(
        'currentPage'
    ) as HTMLInputElement).value =
    this.pdfviewerControl.currentPageNumber.toString();
    this.updatePageNavigation();
}
public onCurrentPageBoxClicked(e: ClickEventArgs): void {
    (document.getElementById('currentPage') as HTMLInputElement).select();
}
public onCurrentPageBoxKeypress(e: KeyboardEvent): boolean {
    if ((e.which < 48 || e.which > 57) && e.which !== 8 && e.which !== 13) {
        e.preventDefault();
        return false;
    } else {
        // tslint:disable-next-line:radix
        const currentPageNumber: number = parseInt(
            (document.getElementById('currentPage') as HTMLInputElement).value
        );
        if (e.which === 13) {
            if (
                currentPageNumber > 0 &&
                currentPageNumber <= this.pdfviewerControl.pageCount
            ) {
                this.pdfviewerControl.navigation.goToPage(currentPageNumber);
            } else {
                // tslint:disable-next-line:max-line-length
                (document.getElementById(
                    'currentPage'
                ) as HTMLInputElement).value =
                this.pdfviewerControl.currentPageNumber.toString();
            }
        }
        return true;
    }
}

```

```

}
private updatePageNavigation(): void {
if(this.pdfviewerControl.currentPageNumber === 1) {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
false
);
this.customToolbar.enableItems(document.getElementById('nextPage'), true);
} else if (
this.pdfviewerControl.currentPageNumber ===
this.pdfviewerControl.pageCount
) {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
true
);
this.customToolbar.enableItems(
document.getElementById('nextPage'),
false
);
} else {
this.customToolbar.enableItems(
document.getElementById('previousPage'),
true
);
this.customToolbar.enableItems(document.getElementById('nextPage'), true);
}
}

private updateZoomButtons(): void {
if(this.pdfviewerControl.zoomPercentage <= 50) {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), true);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), false);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
} else if (this.pdfviewerControl.zoomPercentage >= 400) {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), false);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), true);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
} else {
this.zoomToolbar.enableItems(document.getElementById('zoomIn'), true);
this.zoomToolbar.enableItems(document.getElementById('zoomOut'), true);
this.zoomToolbar.enableItems(document.getElementById('fitPage'), true);
}
}
// tslint:disable-next-line
private readFile(args: any): void {
// tslint:disable-next-line
let uploadedFiles: any = args.target.files;
if(args.target.files[0] !== null) {
let uploadedFile: File = uploadedFiles[0];
if (uploadedFile) {
let reader: FileReader = new FileReader();
reader.readAsDataURL(uploadedFile);
// tslint:disable-next-line
let proxy: any = this;
// tslint:disable-next-line
reader.onload = (e: any): void => {
let uploadedFileUrl: string = e.currentTarget.result;

```



```
proxy.pdfviewerControl.load(uploadedFileUrl, null);  
};  
}  
}  
}
```

Find the sample of [Toolbar Customization](#)

Unload the PDF document programmatically

The PDF Viewer library allows you to unload the PDF document being displayed in the PDF Viewer control programmatically using the [unload\(\)](#) method.

The following steps are used to unload the PDF document programmatically.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: Add the following code snippet to perform the unload operation.

```
`html  
  
<button (click)="unload()">Unload Document</button>  
,  
  
`typescript  
  
unload() {  
  // Unload the PDF document.  
  var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];  
  viewer.unload();  
}  
,
```

Find the Sample, [how to unload the PDF document programmatically](#)

Load PDF documents dynamically

The PDF Viewer library allows to switch or load the PDF documents dynamically after the initial load operation. To achieve this, load the PDF document as a base64 string or file name in PDF Viewer control using the [Load\(\)](#) method dynamically.

The following steps are used to load the PDF document dynamically.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample in Angular.

Step 2: Use the following code snippet to load the document from Base64 string.

```
`html  
  
<button (click)="load_1()">LoadDocumentFromBase64</button>  
,  
  
`typescript  
  
load_1() {  
  var viewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
```

```
viewer.load(
"data:application/pdf;base64,.....",
null
);
}
`
```

Step 3: Use the following code snippet to load PDF document using document name.

```
`html
<button (click)="load_2()">LoadDocumentFromBase64</button>
`

`typescript
load_2() {
// Load PDF document using file name
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
viewer.load('https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf', null);
}
`
```

Find the Sample, [how to load the PDF document dynamically](#)

[View sample in GitHub](#)

Include the Authorization token

The PDF Viewer library allows you to include the authorization token in the PDF viewer AJAX request using the properties of the ajaxRequest header available in AjaxRequestSettings, and it will be included in every AJAX request send from PDF Viewer.

The following steps are used to include the authorization token to the PDF viewer control.

Step 1: Follow the steps provided in the [link](#) to create simple PDF Viewer sample in Angular.

Step 2: Add the following code snippet to include the authorization token.

```
`html
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[ajaxRequestSettings]="ajaxSetting"
style="height:640px;display:block">
</ejs-pdfviewer>
`
```

```
`typescript
public ajaxSetting = {
  ajaxHeaders: [
    {
      headerName: "Authorization",
      headerValue: "Bearer 64565dfgfdsjweiuvbiuyhiueygf"
    }
  ],
  withCredentials: false
};
`
```

Find the Sample [how to include authorization token](#)

Get the Base 64 string of the loaded PDF document

The PDF Viewer library allows you to get the base 64 string of the loaded PDF document by using **saveAsBlob()** method. The entire PDF document will get as blob as like memory stream. So, we can save the blob or convert into stream and we can save it in the database. We can also load the PDF document from base 64 string using the **load()** method.

The following steps are used to get the base 64 string of the loaded PDF document in the PDF viewer control.

Step 1: Follow the steps provided in the [link](#) to create simple PDF Viewer sample in Angular.

Step 2: Add the following code snippet to get the base 64 string with button click event.

```
`html
<button (click)="base64ofloadedDocument()">base64Document</button>
`
```

```
`typescript
base64ofloadedDocument() {
  var viewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
  viewer.saveAsBlob().then(function (value) {
    var data = value;
    var reader = new FileReader();
    reader.readAsDataURL(data);
    reader.onload = () => {
      var base64data = reader.result;
      // get base 64 string.
      console.log(base64data);
    }
  });
}
```

```
};
});
`
```

Step 3: Use the following code snippet inside the **saveAsBlob()** method to load the document from the base 64 string.

```
`typescript
// load the document from base 64 string.
viewer.load(base64data, null);
`
```

Find the Sample, [how to get the Base 64 string of the loaded PDF document](#)

[View sample in GitHub](#)

Customize the selection border

The PDF Viewer library allows you to customize the annotations selection borders using the [annotationSelectorSettings](#) Property.

The following steps are used to customize the selection border.

Step 1: Follow the steps provided in the [link](#) to create simple PDF Viewer sample in Angular.

Step 2: Add the following code snippet in app.component.html and app.component.ts files to customize selection borders.

STANDALONE

```
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document '
[freeTextSettings]="annotationsettings"
[rectangleSettings]="annotationsettings"
[stampSettings]="annotationsettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service '
[documentPath]='document '
[freeTextSettings]="annotationsettings"
[rectangleSettings]="annotationsettings"
[stampSettings]="annotationsettings"
style="height:640px;display:block">
</ejs-pdfviewer>
```

```
`typescript
public annotationsettings: any = {
annotationSelectorSettings: {
```

```
selectionBorderColor: 'yellow',
resizerShape: 'Circle',
selectorLineDashArray: 9
}
};
`
```

Find the Sample [how to customize the selection border](#)

Extract Text

The PDF Viewer library allows you to extract the text from a page along with the bounds. Text extraction can be done using the [isExtractText](#) property and [extractTextCompleted](#) event.

The following steps are used to extract the text from the page.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: The following code snippet explains how to extract the text from a page .

```
`html
<ejs-pdfviewer #pdfViewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
(extractTextCompleted)='extractTextCompleted($event)'
[isExtractText]=true
style="height:640px;display:block">
</ejs-pdfviewer>
`

`typescript
public extractTextCompleted(e: any): void {
// Extract the Complete text of load document
console.log(e);
console.log(e.documentTextCollection[1]);
// Extract the Text data.
console.log(e.documentTextCollection[1][1].TextData);
// Extract Text in the Page.
console.log(e.documentTextCollection[1][1].PageText);
// Extract Text along with Bounds
console.log(e.documentTextCollection[1][1].TextData[0].Bounds);
}
```

Find the Sample, [how to Extract Text](#)

Import and Export annotation as object

The PDF Viewer library allows you to import annotations from objects or streams instead of loading it as a file. To import such annotation objects, the PDF Viewer control must export the PDF annotations as objects by using the [ExportAnnotationsAsObject\(\)](#) method. Only the annotation objects that are exported from the PDF Viewer can be imported.

The following steps are used to import and export annotations as objects.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: Use the following code snippet to perform import and export annotation.

```
`html
<button (click)="export()">Export Annotation</button>
<button (click)="import()">Import Annotation</button>
`
`ts
//export annotation as object.
export (): void {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
viewer.exportAnnotationsAsObject().then(function (value) {
exportObject = value;
});
}
//import annotation that are exported as object.
import(): void {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
viewer.importAnnotation(JSON.parse(exportObject));
}
`
```

Find the Sample, [how to import and export annotation as object](#)

Delete a specific annotation using deleteAnnotationById

The PDF Viewer library allows you to delete a specific annotation from a PDF document. Deleting a specific annotation can be done using the **deleteAnnotationById()** method. This method is used to delete a specific annotation using its id.

The following steps are used to delete a specific annotation from PDF Document.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: Use the following code snippet to delete a specific annotation using `deleteAnnotationById()` method.

```
`html
<button (click)="deleteAnnotationById()">Delete Annotation by Id</button>
`

`typescript
// Delete Annotation by id.
deleteAnnotationById() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
viewer.annotationModule.deleteAnnotationById(
viewer.annotationCollection[0].annotationId
);
}
```

Find the sample [how to delete a specific annotation using deleteAnnotationById](#)

[Open Thumbnail pane programmatically](#)

The PDF Viewer library allows you to open the thumbnail pane programmatically using the [openThumbnailPane\(\)](#) method.

The following steps are used to open the thumbnail.

Step 1: Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

Step 2: Use the following code snippet to open thumbnail.

```
`html
<button (click)="openThumbnail()">Open Thumbnail Pane</button>
`

`ts
openThumbnail() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
// Open Thumbnail pane.
viewer.thumbnailViewModule.openThumbnailPane();
}
```

Find the sample, [how to open the thumbnail pane programmatically](#)

How to enable and disable the delete button based on annotation selection and unselection events

In the Syncfusion PDF viewer, enable and disable the delete button while selecting and unselecting annotations by using the **annotationSelect** and **annotationUnSelect** events.

Here is an example of how you can enable and disable the delete button while selecting and unselecting annotations:

STANDALONE

```
<ejs-pdfviewer #pdfviewer id='pdfViewer'
[documentPath]='document'
[enableToolbar]=false
[enableNavigationToolbar]=false
(annotationSelect)="annotationSelect($event) "
(annotationUnSelect)="annotationUnSelect($event) "
style="height:640px; display: block">
</ejs-pdfviewer>
```

SERVER-BACKED

```
<ejs-pdfviewer #pdfviewer id='pdfViewer'
[serviceUrl]='service'
[documentPath]='document'
[enableToolbar]=false
[enableNavigationToolbar]=false
(annotationSelect)="annotationSelect($event) "
(annotationUnSelect)="annotationUnSelect($event) "
style="height:640px; display: block">
</ejs-pdfviewer>
```

`html

```
<ejs-toolbar id='topToolbar' #customToolbar>
```

```
<e-item
```

```
prefixIcon="e-delete-1"
```

```
tooltipText="Delete annotation"
```

```
id ="DeleteButton"
```

```
disabled="true"
```

```
(click)="deleteSelectedAnnotation()">
```

```
</e-items>
```

```
</ejs-toolbar>
```

```
,
```

`typescript

```
public annotationSelect(e: any): void {
```

```
this.customToolbar.items[1].disabled = false;
```



```

}
public annotationUnSelect(e: any): void {
  this.customToolbar.items[1].disabled = true;
}
public deleteSelectedAnnotation(): void {
  this.pdfviewerControl.annotation.deleteAnnotation();
  this.customToolbar.items[1].disabled = true;
}
`

```

Find the sample [how to enable and disable the delete button while selecting and unselecting annotations](#).

Add the custom stamp based on the free text bounds

To add a custom stamp in the Syncfusion PDF viewer based on the free text bounds, obtain the value of the bound for the free text in the Pixel in the **annotationAdd** event while adding the free text. However, when a custom stamp is added programmatically, the offset values are set in points. So, you must convert the value of the bound for the free text into a point to add the custom stamp to the free text bounds position.

Here is an example of how you can add the custom stamp based on the free text bounds:

```

`typescript
public annotationAdd(args) {
  if (args.annotationType === 'FreeText') {
    this.x = (args.annotationBound.left * 72) / 96;
    this.y = (args.annotationBound.top * 72) / 96;
    console.log(args);
  }
}
`

```

Find the sample [how to add the custom stamp based on the free text bounds](#).

Install packages required for versions below 12

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-pdfviewer@ngcc](#) package to the application.

```

`bash
npm install @syncfusion/ej2-angular-pdfviewer@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-pdfviewer:"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.

Load Microsoft Office files

The PDF Viewer library allows you to load Microsoft office files such as PowerPoint, Excel, Word and image by using the ajax request.

The following steps are used to load the office files in the PDF Viewer.

Step 1: Follow the steps provided in this [link](#) to create simple PDF Viewer sample in Angular.

Step 2: Add the following code to the controller.cs file in the web service project to load the Microsoft office files. In the `GetImageStream()` method, a word document is converted into a PDF document, and return that PDF document into a base64 string. Similarly, load the PowerPoint, Excel and image into the PDF Viewer.

```
`c#
//Post action for loading the Office products.
public IActionResult GetImageStream([FromBody] Dictionary<string, string> jsonObject)
{
    if (jsonObject.ContainsKey("data"))
    {
        string base64 = jsonObject["data"];
        //string fileName = args.FileData[0].Name;
        string type = jsonObject["type"];
        string data = base64.Split(',')[1];
        byte[] bytes = Convert.FromBase64String(data);
        var outputStream = new MemoryStream();
        Syncfusion.Pdf.PdfDocument pdfDocument = new Syncfusion.Pdf.PdfDocument();
        using (Stream stream = new MemoryStream(bytes))
        {
            switch (type)
            {
                case "docx":
                case "dot":
```

```
case "doc":
case "dotx":
case "docm":
case "dotm":
case "rtf":
    Syncfusion.DocIO.DLS.WordDocument doc = new Syncfusion.DocIO.DLS.WordDocument(stream,
    GetWFormatType(type));
    //Initialization of the DocIORenderer for Word to PDF conversion.
    DocIORenderer render = new DocIORenderer();
    //Convert a Word document into a PDF document.
    pdfDocument = render.ConvertToPDF(doc);
    doc.Close();
    break;
case "pptx":
case "pptm":
case "potx":
case "potm":
    //Load or open a PowerPoint Presentation.
    IPresentation pptxDoc = Presentation.Open(stream);
    pdfDocument = PresentationToPdfConverter.Convert(pptxDoc);
    pptxDoc.Close();
    break;
case "xlsx":
case "xls":
    ExcelEngine excelEngine = new ExcelEngine();
    //Load or open an existing workbook through the Open method of IWorkbooks.
    IWorkbook workbook = excelEngine.Excel.Workbooks.Open(stream);
    //Initialize XlsIO renderer.
    XlsIORenderer renderer = new XlsIORenderer();
    //Convert an Excel document into a PDF document.
    pdfDocument = renderer.ConvertToPDF(workbook);
    workbook.Close();
    break;
```

```

case "jpeg":
case "jpg":
case "png":
case "bmp":
//Add a page to the document.
PdfPage page = pdfDocument.Pages.Add();
//Create PDF graphics for the page.
PdfGraphics graphics = page.Graphics;
PdfBitmap image = new PdfBitmap(stream);
//Draw the image.
graphics.DrawImage(image, 0, 0);
break;
}
}
pdfDocument.Save(outputStream);
outputStream.Position = 0;
byte[] byteArray= outputStream.ToArray();
pdfDocument.Close();
outputStream.Close();
string base64String = Convert.ToBase64String(byteArray);
return Content("data:application/pdf;base64," + base64String);
}
return Content("data:application/pdf;base64," + "");
}
`

```

Step 3: In the following code, an XMLHttpRequest will generate the responseText from the base64 string and set that text as the `documentPath` of the PDF Viewer.

```

`html
function readURL(li, args) {
var file = args.rawFile;
var reader = new FileReader();
var type = args.type;
reader.addEventListener('load', function () {

```

```

let post = JSON.stringify({
  'data': reader.result,
  'type': type
})
const url = "https://localhost:44327/pdfviewer/GetImageStream"
let xhr = new XMLHttpRequest()
xhr.open('Post', url, true)
xhr.setRequestHeader('Content-type', 'application/json; charset=UTF-8')
xhr.send(post);
xhr.onload = function (args) {
  viewer = document.getElementById('pdfViewer').ej2_instances[0];
  viewer.documentPath = this.responseText;
}
},
false
);
if (file) {
  reader.readAsDataURL(file);
}
}
,

```

[View sample in GitHub.](#)

How to Change the Font Family in the Type Signature

Change the font family in the Type Signature of the Syncfusion PDF Viewer by adding a custom CSS stylesheet to the document, and then apply the desired font family to the Type Signature element. Include the Google Font link in the HTML head section to apply the Google Font.

use the `handWrittenSignatureSettings` property of the PdfViewer component and modify the `fontFamily` property.

```

`html
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Allura" >
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Tangerine">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Sacramento">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Inspiration">
,

```

```

`ts
changeFontFamily() {
var pdfviewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfviewer.handWrittenSignatureSettings.typeSignatureFonts = [
'Allura',
'Tangerine',
'Sacramento',
'Inspiration',
];
}
`

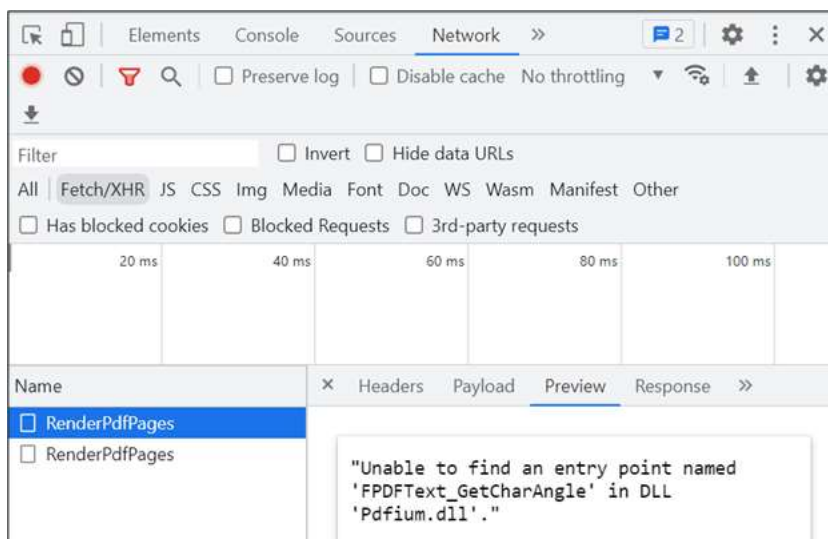
```

Find the sample [how to Change the Font Family in the Type Signature](#)

Resolve "Unable to find an entry point named FPDFText_GetCharAngle" error

From the release of version **21.1.0.35 (2023 Volume 1)** of Essential Studio, the Pdfium package has been upgraded to improve various functionalities like text search, text selection, rendering, and even performance. If you are updating your project to this version of the Syncfusion PDF Viewer, you may encounter the **"Web-Service is not listening"** error. The Network tab can help you identify the root cause of the issue, which is typically caused by an old version of pdfium assembly being referenced in the local web service project. Below are the assemblies to be referred to in the respective operating systems.

- Windows – pdfium.dll
- Linux – libpdfium.so
- OSX – libpdfium.dylib



To solve this issue, you should follow the below steps:

1. Clear the bin and object files of the web-service project.
2. Re-publish the web-service project.

Note: Note: If you are hosting your application in Azure, AWS, or in Linux environments, delete the older published files and republish the application.

How to clear the "Web-service is not listening" to error

PDF Viewer

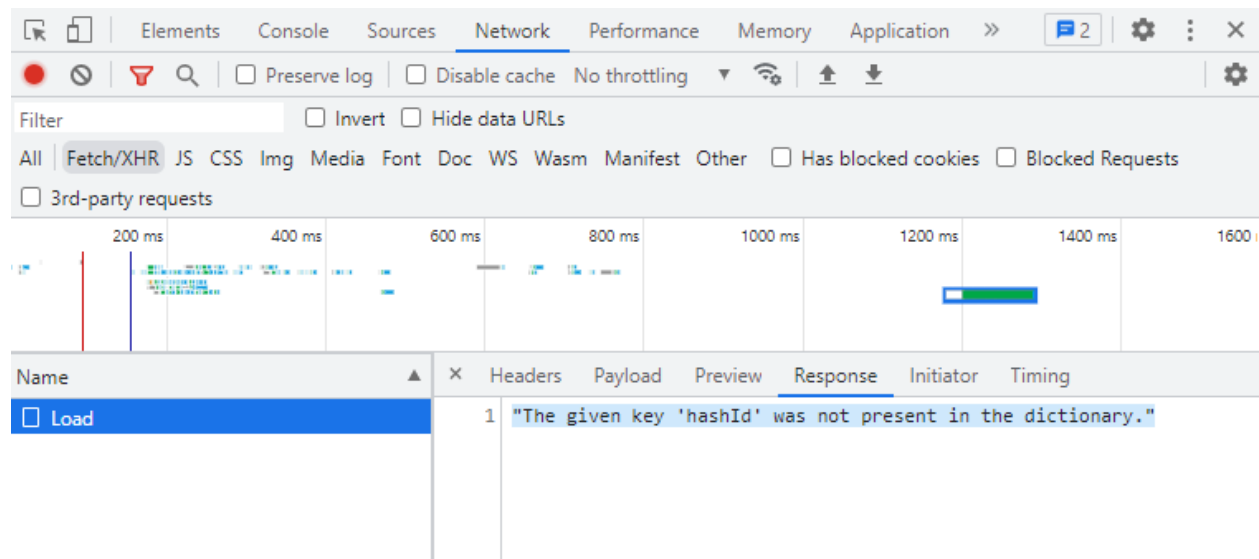


Web-service is not listening. PDF Viewer depends on web-service for all it's features. Please start the web service to continue.

OK

If you are facing a **Web-service is not listening** to error in the Syncfusion PDF Viewer, there could be several reasons for this. To troubleshoot the issue, you can use the Network tab in your browser's developer tools to gather more information. Here are the steps you can follow:

Step 1: Open the browser's developer tools by right-clicking on the page and selecting **Inspect** from the dropdown menu. Then Navigate to the **Network** tab. This will show you all of the requests that are being made by the page.



Step 2: Try to request the web service. If the service is not listening, the request will fail, and you should see an error message in the Network tab. Click on the failing request to see the details of the error, such as the error message or stack trace. This can help you identify the root cause of the issue. Check the server logs for any errors or warnings that may indicate the cause of the issue and help you to troubleshoot the problem.

Step 3: Check the request URL and parameters to see if they are correct. If there is a type or an incorrect parameter, the web service may be unable to process the request.

By following these steps and using the Network tab in your browser's developer tools, you can gather more information about the issue and troubleshoot the problem more effectively.

Note: Make sure you are connected to the internet and that your connection is stable. You can try accessing other websites or services to see if they are working, and make sure the URL you are using to access the web service is correct and properly formatted.

Here are some common exceptions

- File not found.
- Document cache not found.
- Document pointer does not exist in the cache.

File not found

If you are encountering an error message stating that the web service is not listening due to a file not being found in the Syncfusion PDF viewer, you can try the following steps to resolve the issue:

Check the file path

Ensure that the file path you use to access the PDF file is correct and that the file exists in that location. You will need to update the file path if the file does not exist.

Document cache not found

The **Document cache not found** exception in Syncfusion PDF Viewer typically occurs when the cache used to store the rendered pages of a PDF document is not found or has been deleted. This can happen if the cache directory is changed or deleted or if the application is running in a different environment than it was previously.

Check for multiple instances

It's possible that you have multiple instances of the Syncfusion PDF Viewer running simultaneously, which can cause issues with the document cache. To check for this, open the Task Manager on your computer and look for any instances of the Syncfusion PDF Viewer running. If you find multiple instances, try closing them all and reopening the viewer.

We can use Redis cache and distributive cache for this issue.

Check your network connection

Ensure that your network connection is stable and strong enough to support the web service you are trying to use. Sometimes, simply restarting the web service can resolve the issue. Try stopping and starting the service again to see if it resolves the problem.

The document pointer does not exist in the cache.

The **Document pointer does not exist in the cache** exception in the Syncfusion PDF Viewer usually occurs when there is an issue with loading or caching the PDF document. This error can be caused by a variety of reasons, including:

To clear this error in the Syncfusion PDF Viewer, you can try the following steps:

Step 1: Clearing the cache may help resolve the issue. To clear the cache, navigate to the cache location, which can be found in the Syncfusion PDF Viewer's settings or configuration files. Once you locate the cache folder, delete its contents.

Step 2: Try reloading the document to ensure it is loaded correctly. You can do this by calling the controller's Load() method. Ensure the document is not already loaded before attempting to load it again.

Step 3: Restart the application. If clearing the cache does not work, you can try restarting the PDF Viewer application. This will reload all the necessary components and may resolve the error.

Internal server error

Server-side exceptions happen for various use cases. We can't just define them if they are document-specific, provide the document, or you may need to contact Syncfusion support for further assistance.

Load N number of pages on initial loading

The initial rendering in a PDF viewer allows users to control the number of pages displayed when opening a PDF document. This improves the user experience by providing flexibility in loading a specific number of pages initially, while additional pages are dynamically rendered as the user scrolls through the document. This approach enhances the responsiveness of the PDF viewer and minimizes delays as users can access specific pages without waiting for the entire document to load.

To utilize this capability in Syncfusion PDF Viewer, use the [initialRenderPages](#) property. You can achieve the desired outcome by setting this property to the desired number of pages you want to load initially. However, it's important to exercise caution when setting a high value for the initialRenderPages in large documents with numerous pages. Rendering a large number of pages simultaneously can increase memory usage and slow down loading times, impacting the performance of the PDF viewer.

Using the `initialRenderPages` property judiciously is advisable, especially when dealing with larger documents. It is more suitable for scenarios where a smaller range of pages, such as 10-20, can be loaded to provide a quick initial view of the document.

STANDALONE

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[initialRenderPages]='initialRender'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
```

```
PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
  succinctly.pdf';
  public initialRender = 10;
  ngOnInit(): void {
  }
}
```

SERVER-BACKED

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService
} from '@syncfusion/ej2-angular-pdfviewer';
@Component({
  selector: 'app-root',
  // specifies the template string for the PDF Viewer component
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
[initialRenderPages]='initialRender'
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [ LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
  succinctly.pdf';
  public initialRender = 10;
  ngOnInit(): void {
  }
}
```

Find the sample [how to load n number of pages on initial loading](#)

Retry Timeout

The **Retry Timeout** feature allows you to specify a duration for the PDF Viewer to retry failed AJAX requests before considering them permanent failures. It helps in handling temporary failures due to network issues or server unavailability.

The `retryTimeout` allows developers to specify a duration after which the AJAX request should retry failed requests automatically. Developers can ensure a more resilient and fault-tolerant PDF viewing experience by configuring an appropriate retry timeout value.

By default, when an AJAX request fails, the Retry Timeout property is set to 0, indicating that no timeout is set. In this case, the PDF Viewer will wait indefinitely for a response, potentially leading to a hanging request. However, you can set the Retry Timeout property to a positive number, specifying the maximum number of seconds the PDF Viewer should wait for a response. If the response is not received within the specified time, the request will be aborted, triggering an appropriate error or timeout property.

To set the retry timeout, use the `retryTimeout` property in the PDF Viewer configuration. This property takes a value in seconds.

```
`html
<ejs-pdfviewer
id="pdfViewer"
[serviceUrl]="service"
[documentPath]="document"
[retryTimeout]="10"
[retryCount]="5"
style="height:640px;display:block">
</ejs-pdfviewer>
`
```

In the given example, the `retryTimeout` is set to 10 seconds, and the `retryCount` is set to 5. This means that if a request made by the PDF Viewer takes longer than 10 seconds to receive a response, it will be considered a timeout. In such cases, The PDF Viewer will resend the same request based on the `retryCount`. Here, this process will repeat up to maximum of 5 retries.

When an exception occurs during the AJAX request in the context of the PDF Viewer, the request will wait for the specified `retryTimeout` duration. If the timeout duration is exceeded, the PDF Viewer will decrement the `retryCount` and attempt to load the document again. This retry process continues until the document is successfully loaded or the `retryCount` limit is reached.

The `retryCount` property of the PDF Viewer allows you to set the number of retries for a specific request. This feature is particularly useful for handling temporary errors such as network timeouts or server issues. By initiating new requests according to the retry count, ensure a smoother user experience and efficiently handle network or server problems.

If the timeout duration specified by `retryTimeout` is exceeded during the AJAX request, the PDF Viewer will decrement the `retryCount` and initiate a new request. This process will continue until the document is successfully loaded or the retry count limit is reached. This functionality helps improve the handling of temporary errors and ensures a more efficient and user-friendly experience.

Find the sample [Retry Timeout](#)

Configure Redis Cache

Redis is an open-source, in-memory data structure store that is often used as a cache, message broker, and database. `Redis cache` is a key-value data store that stores data in memory, which makes it very

fast and efficient. The data can be stored and retrieved quickly without the need for disk access, which makes Redis cache ideal for applications that require fast access to data.

Redis can be used to improve the performance of the PDF Viewer by caching frequently accessed PDF documents and reducing the number of requests to the server. Redis is an in-memory cache, so data stored in Redis should be considered temporary. In case of cache eviction or server restart, the data will be lost, and you would need to fetch it again from the primary data source.

To configure Redis, you will need to follow these steps

Step 1: Create Redis cache refer to this [link](#)

Step 2: Create a PDFViewer web service application for that use the below link for reference.

[Refer to this link](#)

or you can get the sample web service from GitHub [link](#).

Step 3: Install Redis Cache package

You need to install the `StackExchange.Redis` package and `Microsoft.Extensions.Caching.Redis` using the NuGet Package Manager

Step 4: Configure Redis Cache

In the `ConfigureServices` method of the `Startup` class, you need to add the Redis Cache service using the `AddDistributedRedisCache()`

method. You also need to provide the Redis Cache connection string.

```
`cs
public void ConfigureServices(IServiceCollection services)
{
    // Add Redis Cache
    services.AddDistributedRedisCache(options =>
    {
        options.Configuration = redisCacheConnectionString;
    });
}
```

Step 5: Use the Redis cache in the PDF Viewer controller action:

To use Redis Cache in PDF Viewer, you can implement the `IDistributedCache` interface and use the Redis Cache service to store and

retrieve

the PDF document bytes.

```
`cs
private readonly IHostingEnvironment _hostingEnvironment;
```

```

public IMemoryCache _cache;
private IDistributedCache _dCache;
private IConfiguration _configuration;
private int _slidingTime = 0;
string path;

public PdfViewerController(IMemoryCache memoryCache, IHostingEnvironment hostingEnvironment,
IDistributedCache cache, IConfiguration configuration)
{
    _cache = memoryCache;
    _dCache = cache;
    _hostingEnvironment = hostingEnvironment;
    _configuration = configuration;
    path = configuration["DOCUMENTPATH"];
    //check the document path environment variable value and assign default data folder
    //if it is null.
    path = string.IsNullOrEmpty(path) ? Path.Combine(hostingEnvironment.ContentRootPath, "Data") :
    Path.Combine(hostingEnvironment.ContentRootPath, path);
}

```

Once Redis is configured, the Syncfusion PDF Viewer application will automatically use Redis cache for improved performance and scalability.

Customize toolbar in PDF Viewer component

How to customize existing toolbar in PDF Viewer component

PDF Viewer allows you to customize(add, show, hide, enable, and disable) existing items in a toolbar.

- Add - New items can be defined by [CustomToolbarItemModel](#) and with existing items in [ToolbarSettings](#) property. Newly added item click action can be defined in [toolbarclick](#).
- Show, Hide - Existing items can be shown or hidden using the [ToolbarSettings](#) property. Pre-defined toolbar items are available with [ToolbarItem](#).
- Enable, Disable - Toolbar items can be enabled or disabled using [enabletoolbaritem](#)

STANDALONE

```

import { Component, OnInit } from '@angular/core';
import {
    LinkAnnotationService, BookmarkViewService, MagnificationService,
    ThumbnailViewService, ToolbarService, NavigationService,
    AnnotationService, TextSearchService, TextSelectionService,
    PrintService, FormDesignerService, FormFieldsService, CustomToolbarItemModel
} from '@syncfusion/ej2-angular-pdfviewer';
import { ComboBox } from '@syncfusion/ej2-dropdowns';

```

```

import { TextBox } from "@syncfusion/ej2-inputs";
@Component({
  selector: 'app-root',
  template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]="document"
[resourceUrl]="resource"
[toolbarSettings]="toolbarSettings"
(toolbarClick)="toolbarClick($event)"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
  providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource = 'https://cdn.syncfusion.com/ej2/24.1.41/dist/ej2-pdfviewer-
lib';
  public toolItem1: CustomToolbarItemModel = {
    prefixIcon: 'e-icons e-paste',
    id: 'print',
    tooltipText: 'Custom toolbar item',
  };
  public toolItem2: CustomToolbarItemModel = {
    id: 'download',
    text: 'Save',
    tooltipText: 'Custom toolbar item',
    align: 'right'
  };
  LanguageList: string[] = ['Typescript', 'Javascript', 'Angular', 'C#', 'C',
'Python'];
  public toolItem3: CustomToolbarItemModel = {
    type: 'Input',
    tooltipText: 'Language List',
    cssClass: 'percentage',
    align: 'Left',
    id: 'dropdown',
    template: new ComboBox({ width: 100, value: 'TypeScript', dataSource:
this.LanguageList, popupWidth: 85, showClearButton: false, readonly: false })
  };
  public toolItem4: CustomToolbarItemModel = {
    type: 'Input',
    tooltipText: 'Text',
    align: 'Right',
    cssClass: 'find',
    id: 'textbox',
    template: new TextBox({ width: 125, placeholder: 'Type Here', created:
this.OnCreateSearch})
  }
  public toolbarSettings = {
    showTooltip: true,
    toolbarItems: [this.toolItem1, this.toolItem2, 'OpenOption',
'PageNavigationTool', 'MagnificationTool', this.toolItem3, 'PanTool',

```

```

'SelectionTool', 'SearchOption', 'PrintOption', 'DownloadOption',
'UndoRedoTool', 'AnnotationEditTool', 'FormDesignerEditTool', this.toolItem4,
'CommentTool', 'SubmitForm']
};
public toolbarClick(args: any): void {
var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
if (args.item && args.item.id === 'print') {
pdfViewer.printModule.print();
} else if (args.item && args.item.id === 'download') {
pdfViewer.download();
}
}
public OnCreateSearch(this: any): any {
this.addIcon('prepend', 'e-icons e-search');
}
ngOnInit(): void {
}
}

```

SERVER-BACKED

```

import { Component, OnInit } from '@angular/core';
import {
LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService, CustomToolbarItemModel
} from '@syncfusion/ej2-angular-pdfviewer';
import { ComboBox } from "@syncfusion/ej2-dropdowns";
import { TextBox } from "@syncfusion/ej2-inputs";
@Component({
selector: 'app-root',
template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]="document"
[serviceUrl]="service"
[toolbarSettings]="toolbarSettings"
(toolbarClick)="toolbarClick($event)"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
providers: [LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService]
})
export class AppComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
public service =
'https://services.syncfusion.com/angular/production/api/pdfviewer';
public toolItem1: CustomToolbarItemModel = {
prefixIcon: 'e-icons e-paste',
id: 'print',
tooltipText: 'Custom toolbar item',
};
}

```

```

public toolItem2: CustomToolbarItemModel = {
  id: 'download',
  text: 'Save',
  tooltipText: 'Custom toolbar item',
  align: 'right'
};
LanguageList: string[] = ['Typescript', 'Javascript', 'Angular', 'C#', 'C', 'Python'];
public toolItem3: CustomToolbarItemModel = {
  type: 'Input',
  tooltipText: 'Language List',
  cssClass: 'percentage',
  align: 'Left',
  id: 'dropdown',
  template: new ComboBox({ width: 100, value: 'TypeScript', dataSource:
this.LanguageList, popupWidth: 85, showClearButton: false, readonly: false })
};
public toolItem4: CustomToolbarItemModel = {
  type: 'Input',
  tooltipText: 'Text',
  align: 'Right',
  cssClass: 'find',
  id: 'textbox',
  template: new TextBox({ width: 125, placeholder: 'Type Here', created:
this.OnCreateSearch})
}
public toolbarSettings = {
  showTooltip: true,
  toolbarItems: [this.toolItem1, this.toolItem2, 'OpenOption',
'PageNavigationTool', 'MagnificationTool', this.toolItem3, 'PanTool',
'SelectionTool', 'SearchOption', 'PrintOption', 'DownloadOption',
'UndoRedoTool', 'AnnotationEditTool', 'FormDesignerEditTool', this.toolItem4,
'CommentTool', 'SubmitForm']
};
public toolbarClick(args: any): void {
  var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  if (args.item && args.item.id === 'print') {
    pdfViewer.printModule.print();
  } else if (args.item && args.item.id === 'download') {
    pdfViewer.download();
  }
}
public OnCreateSearch(this: any): any {
  this.addIcon('prepend', 'e-icons e-search');
}
ngOnInit(): void {
}
}

```

Note: Default value of toolbar items is ['OpenOption', 'PageNavigationTool', 'MagnificationTool', 'PanTool', 'SelectionTool', 'SearchOption', 'PrintOption', 'DownloadOption', 'UndoRedoTool', 'AnnotationEditTool', 'FormDesignerEditTool', 'CommentTool', 'SubmitForm']

Align Property

The align property is used to specify the alignment of a toolbar item within the toolbar.

Left: Aligns the item to the left side of the toolbar.

Right: Aligns the item to the right side of the toolbar.

Tooltip Property

The tooltip property is used to set the tooltip text for a toolbar item. Tooltip provides additional information when a user hovers over the item.

CssClass Property

The cssClass property is used to apply custom CSS classes to a toolbar item. It allows custom styling of the toolbar item.

Prefix Property

The prefix property is used to set the CSS class or icon that should be added as a prefix to the existing content of the toolbar item.

ID Property

The id property within a CustomToolbarItemModel is a compulsory attribute that plays a vital role in toolbar customization. It serves as a unique identifier for each toolbar item, facilitating distinct references and interactions.

When defining or customizing toolbar items, it is mandatory to assign a specific and descriptive id to each item.

These properties are commonly used when defining custom toolbar items with the CustomToolbarItemModel in the context of Syncfusion PDF Viewer. When configuring the toolbar using the ToolbarSettings property, you can include these properties to customize the appearance and behavior of each toolbar item.

Note: When customizing toolbar items, you have the flexibility to include either icons or text based on your design preference.

[View sample in GitHub](#)

Know the supported conformance PDF documents in Anglar PDF Viewer component

The PDF Viewer supports the below conformance documents:

- **PDF/A-1a conformance**
- **PDF/A-1b conformance**
- **PDF/X-1a conformance**
- **PDF/A-2a conformance**
- **PDF/A-2b conformance**
- **PDF/A-2u conformance**
- **PDF/A-3a conformance**
- **PDF/A-3b conformance**
- **PDF/A-3u conformance**
- **PDF/A-4 conformance**
- **PDF/A-4e conformance**
- **PDF/A-4f conformance**

Organize Pages Feature

Introduction

Welcome to the User Guide for the Organize Pages feature in JS2 PDF Viewer. This powerful feature allows you to manage your PDF documents efficiently by organizing pages seamlessly. Whether you need to add new pages, remove unnecessary ones, or adjust page orientation, this feature has got you covered.

Getting Started

To access the Organize Pages feature, simply open the PDF document in the JS2 PDF Viewer and navigate to the toolbar. Look for the **Organize Pages** option to begin utilizing these capabilities.

Key Functionalities

- **Add New Pages:** Easily integrate additional content by adding new pages to your document. Simply select the option to insert new pages and customize them as needed.
- **Remove Pages:** Streamline your document management process by removing unnecessary pages with ease. Select the pages you wish to delete and confirm to remove them from the document.
- **Rotate Pages:** Resolve orientation issues by rotating pages clockwise or counterclockwise as required. This feature ensures that your document displays correctly, maintaining clarity and readability.
- **Select All Pages:** Make uniform adjustments and modifications by conveniently selecting all pages at once. This allows for efficient editing and formatting across the entire document.
- **Real-Time Updates:** Experience instant updates as any changes made to the page organization are instantly reflected within the PDF Viewer. Simply click the **Save** button to ensure that your modifications are preserved.
- **Save As Feature:** Preserve your edits by utilizing the **Save As** feature. This allows you to download the modified version of the PDF document for future reference, ensuring that your changes are stored securely.

API's supported

enableOrganizePdf: This API enables or disables the page organizer feature in the PDF Viewer. By default, it is set to **true**, indicating that the page organizer is enabled.

isPageOrganizerOpen: This API determines whether the page organizer dialog will be displayed automatically when a document is loaded into the PDF Viewer. By default, it is set to **false**, meaning the dialog is not displayed initially.

pageOrganizerSettings: This API allows control over various page management functionalities within the PDF Viewer. It includes options to enable or disable actions such as deleting, inserting, rotating, and moving pages. By default, all these actions are enabled.

showPageOrganizerTool: This API controls the visibility of the page organizer tool within the PDF Viewer. If set to **false**, the tool is hidden by default.

openPageOrganizer: This API opens the page organizer dialog within the PDF Viewer, providing access to manage PDF pages.

closePageOrganizer: This API closes the currently open page organizer dialog within the PDF Viewer, if it is present. It allows users to dismiss the dialog when done with page organization tasks.

Conclusion

With the Organize Pages feature in JS2 PDF Viewer, managing your PDF documents has never been easier. Whether you're adding new content, adjusting page orientation, or removing unnecessary pages, this feature provides the tools you need to streamline your document management workflow. Explore these capabilities today and take control of your PDF documents with ease!

[View sample in GitHub](#)

Customize context menu

PDF Viewer allows you to add custom option in context menu. It can be achieved by using the `addCustomMenu()` method and custom action is defined using the `customContextMenuSelect()` method.

Add Custom Option

The following code shows how to add custom option in context menu.

```
`ts
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/23.2.6/dist/ej2-pdfviewer-lib";
ngOnInit(): void {
  // ngOnInit function
}
public menuItems: MenuItemModel[] = [
  {
    text: 'Search In Google',
    id: 'searchingoogle',
    iconCss: 'e-icons e-search'
  },
  {
    text: 'Lock Annotation',
    iconCss: 'e-icons e-lock',
    id: 'lock_annotation'
  },
  {
    text: 'Unlock Annotation',
    iconCss: 'e-icons e-unlock',
    id: 'unlock_annotation'
  },
  {
```

```

text: 'Lock Form Fields',
iconCss: 'e-icons e-lock',
id: 'readonlytrue'
},
{
text: 'Unlock Form Fields',
iconCss: 'e-icons e-unlock',
id: 'readonlyfalse'
},
]

public documentLoaded(e: any): void {
var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfViewer.addCustomMenu(this.menuItems, false, false);
}
`

```

Customize custom option in context menu

The PDF Viewer feature enables customization of custom options and the ability to toggle the display of the default context menu. When the `addCustomMenu` parameter is set to `true`, the default menu is hidden; conversely, when it is set to `false`, the default menu items are displayed.

```

`js
public document: string = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
public resource: string = "https://cdn.syncfusion.com/ej2/23.2.6/dist/ej2-pdfviewer-lib";
ngOnInit(): void {
// ngOnInit function
}

public menuItems: MenuItemModel[] = [
{
text: 'Search In Google',
id: 'searchingoogle',
iconCss: 'e-icons e-search'
},
{
text: 'Lock Annotation',
iconCss: 'e-icons e-lock',

```

```

id: 'lock_annotation'
},
{
text: 'Unlock Annotation',
iconCss: 'e-icons e-unlock',
id: 'unlock_annotation'
},
{
text: 'Lock Form Fields',
iconCss: 'e-icons e-lock',
id: 'readonlytrue'
},
{
text: 'Unlock Form Fields',
iconCss: 'e-icons e-unlock',
id: 'readonlyfalse'
},
]

public documentLoaded(e: any): void {
var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
pdfViewer.addCustomMenu(this.menuItems, false, false);
}
`

```

[Customize added context menu items](#)

The following code shows how to hide/show added custom option in context menu using the `customContextMenuBeforeOpen()` method.

```

`js
export class CustomContextMenuComponent implements OnInit {
public document = 'https://cdn.syncfusion.com/content/pdf/pdf-succinctly.pdf';
public resource: string = 'https://cdn.syncfusion.com/ej2/24.1.41/dist/ej2-pdfviewer-lib';
;
ngOnInit(): void {
}
public menuItems: MenuItemModel[] = [

```

```
{
  text: 'Search In Google',
  id: 'searchingoogle',
  iconCss: 'e-icons e-search'
},
{
  text: 'Lock Annotation',
  iconCss: 'e-icons e-lock',
  id: 'lock_annotation'
},
{
  text: 'Unlock Annotation',
  iconCss: 'e-icons e-unlock',
  id: 'unlock_annotation'
},
{
  text: 'Lock Form Fields',
  iconCss: 'e-icons e-lock',
  id: 'readonlytrue'
},
{
  text: 'Unlock Form Fields',
  iconCss: 'e-icons e-unlock',
  id: 'readonlyfalse'
},
]

public documentLoaded(e: any): void {
  var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  pdfViewer.addCustomMenu(this.menuItems, false, false);
}

public customContextMenuSelect = (e: any): void => {
  var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
  switch (e.id) {
```

```
case 'searchingoogle':
for (var i = 0; i < pdfViewer.textSelectionModule.selectionRangeArray.length; i++) {
var content = pdfViewer.textSelectionModule.selectionRangeArray[i].textContent;
if ((pdfViewer.textSelectionModule.isTextSelection) && (/S/.test(content))) {
window.open('http://google.com/search?q=' + content);
}
}
break;
case 'lock_annotation':
this.lockAnnotations(e);
break;
case 'unlock_annotation':
this.unlockAnnotations(e);
break;
case 'readonlytrue':
this.setReadOnlyTrue(e);
break;
case 'readonlyfalse':
this.setReadOnlyFalse(e);
break;
case 'formfield properties':
break;
default:
break;
}
}

public customContextMenuBeforeOpen = (e: any): void => {
var pdfViewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
for (var i = 0; i < e.ids.length; i++) {
var search = document.getElementById(e.ids[i]);
if (search) {
search.style.display = 'none';
}
```

```

if (e.ids[i] === 'searchingoogle' && (pdfViewer.textSelectionModule) &&
pdfViewer.textSelectionModule.isTextSelection) {
search.style.display = 'block';
} else if (e.ids[i] === "lockannotation" || e.ids[i] === "unlockannotation") {
var isLockOption = e.ids[i] === "lock_annotation";
for (var j = 0; j < pdfViewer.selectedItems.annotations.length; j++) {
var selectedAnnotation: any = pdfViewer.selectedItems.annotations[j];
if (selectedAnnotation && selectedAnnotation.annotationSettings) {
var shouldDisplay = (isLockOption && !selectedAnnotation.annotationSettings.isLock) ||
(!isLockOption && selectedAnnotation.annotationSettings.isLock);
search.style.display = shouldDisplay ? 'block' : 'none';
}
}
} else if ((e.ids[i] === "readonlytrue" || e.ids[i] === "readonlyfalse") &&
pdfViewer.selectedItems.formFields.length !== 0) {
var isReadOnlyOption = e.ids[i] === "readonlytrue";
for (var j = 0; j < pdfViewer.selectedItems.formFields.length; j++) {
var selectedFormFields = pdfViewer.selectedItems.formFields[j];
if (selectedFormFields) {
var selectedFormField = pdfViewer.selectedItems.formFields[j].isReadOnly;
var displayMenu = (isReadOnlyOption && !selectedFormField) || (!isReadOnlyOption &&
selectedFormField);
search.style.display = displayMenu ? 'block' : 'none';
}
}
} else if (e.ids[i] === 'formfield properties' && pdfViewer.selectedItems.formFields.length !== 0) {
search.style.display = 'block';
}
}
}
}
}
}

```

The following is the output of custom context menu with customization.

APP.COMPONENT.TS

```

import { Component, OnInit, ViewEncapsulation, ViewChild } from
'@angular/core';
import {
  LinkAnnotationService,
  BookmarkViewService,
  MagnificationService,
  ThumbnailViewService,
  ToolbarService,
  NavigationService,
  AnnotationService,
  TextSearchService,
  TextSelectionService,
  FormFieldsService,
  FormDesignerService,
  PrintService,
  PageOrganizerService
} from '@syncfusion/ej2-angular-pdfviewer';
import { MenuItemModel } from '@syncfusion/ej2-angular-navigations';
import { CheckBoxComponent } from '@syncfusion/ej2-angular-buttons';
@Component({
  selector: 'app-container',
  template: `<div class="content-wrapper">
    <div>
      <ejs-checkbox label="Hide Default Context Menu" id="hide" #hide
[checked]="false" (change)="contextmenuHelper($event)"></ejs-checkbox>
      <ejs-checkbox label="Add Custom option at bottom" id="toolbar" #toolbar
[checked]="false" (change)="contextmenuHelper($event)"></ejs-checkbox>
    </div>
    <!--Render PDF Viewer component-->
    <ejs-pdfviewer id="pdfViewer"
      [documentPath]='document'
      [resourceUrl]='resource'
      (documentLoad)='documentLoaded($event)'
      (customContextMenuBeforeOpen)='customContextMenuBeforeOpen($event)'
      (customContextMenuSelect)='customContextMenuSelect($event)'
      style="height:640px;display:block">
    </ejs-pdfviewer>
  </div>`,
  encapsulation: ViewEncapsulation.None,
  providers: [
    LinkAnnotationService,
    BookmarkViewService,
    MagnificationService,
    ThumbnailViewService,
    ToolbarService,
    NavigationService,
    TextSearchService,
    TextSelectionService,
    PrintService,
    AnnotationService,
    FormFieldsService,
    FormDesignerService,
    PageOrganizerService
  ],
})

```

```

export class AppComponent implements OnInit {
  @ViewChild('hide')
  public hideObj?: CheckBoxComponent;
  @ViewChild('toolbar')
  public toolbarObj?: CheckBoxComponent;
  public hide: any;
  public toolbar: any;
  public document = 'https://cdn.syncfusion.com/content/pdf/pdf-
succinctly.pdf';
  public resource: string = 'https://cdn.syncfusion.com/ej2/25.1.35/dist/ej2-
pdfviewer-lib';
  ;
  ngOnInit(): void {
  }
  public menuItems: MenuItemModel[] = [
    {
      text: 'Search In Google',
      id: 'search_in_google',
      iconCss: 'e-icons e-search'
    },
    {
      text: 'Lock Annotation',
      iconCss: 'e-icons e-lock',
      id: 'lock_annotation'
    },
    {
      text: 'Unlock Annotation',
      iconCss: 'e-icons e-unlock',
      id: 'unlock_annotation'
    },
    {
      text: 'Lock Form Fields',
      iconCss: 'e-icons e-lock',
      id: 'read_only_true'
    },
    {
      text: 'Unlock Form Fields',
      iconCss: 'e-icons e-unlock',
      id: 'read_only_false'
    },
  ],
  ]
  public documentLoaded(e: any): void {
    var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
    pdfViewer.addCustomMenu(this.menuItems, false, false);
  }
  public customContextMenuSelect = (e: any): void => {
    var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
    switch (e.id) {
      case 'search_in_google':
        for (var i = 0; i <
pdfViewer.textSelectionModule.selectionRangeArray.length; i++) {
          var content =
pdfViewer.textSelectionModule.selectionRangeArray[i].textContent;
          if ((pdfViewer.textSelectionModule.isTextSelection) &&
(/\\S/.test(content))) {

```

```

        window.open('http://google.com/search?q=' + content);
    }
}
break;
case 'lock_annotation':
    this.lockAnnotations(e);
    break;
case 'unlock_annotation':
    this.unlockAnnotations(e);
    break;
case 'read_only_true':
    this.setReadOnlyTrue(e);
    break;
case 'read_only_false':
    this.setReadOnlyFalse(e);
    break;
case 'formfield properties':
    break;
default:
    break;
}
}
}
public customContextMenuBeforeOpen = (e: any): void => {
    var pdfViewer =
    (<any>document.getElementById('pdfViewer')).ej2_instances[0];
    for (var i = 0; i < e.ids.length; i++) {
        var search = document.getElementById(e.ids[i]);
        if (search) {
            search.style.display = 'none';
            if (e.ids[i] === 'search_in_google' &&
            (pdfViewer.textSelectionModule) &&
            pdfViewer.textSelectionModule.isTextSelection) {
                search.style.display = 'block';
            } else if (e.ids[i] === "lock_annotation" || e.ids[i] ===
            "unlock_annotation") {
                var isLockOption = e.ids[i] === "lock_annotation";
                for (var j = 0; j < pdfViewer.selectedItems.annotations.length;
                j++) {
                    var selectedAnnotation: any =
                    pdfViewer.selectedItems.annotations[j];
                    if (selectedAnnotation && selectedAnnotation.annotationSettings)
                    {
                        var shouldDisplay = (isLockOption &&
                        !selectedAnnotation.annotationSettings.isLock) ||
                        (!isLockOption &&
                        selectedAnnotation.annotationSettings.isLock);
                        search.style.display = shouldDisplay ? 'block' : 'none';
                    }
                }
            } else if ((e.ids[i] === "read_only_true" || e.ids[i] ===
            "read_only_false") && pdfViewer.selectedItems.formFields.length !== 0) {
                var isReadOnlyOption = e.ids[i] === "read_only_true";
                for (var j = 0; j < pdfViewer.selectedItems.formFields.length; j++)
                {
                    var selectedFormFields = pdfViewer.selectedItems.formFields[j];
                    if (selectedFormFields) {

```

```

        var selectedFormField =
pdfViewer.selectedItems.formFields[j].isReadOnly;
        var displayMenu = (isReadOnlyOption && !selectedFormField) ||
(!isReadOnlyOption && selectedFormField);
        search.style.display = displayMenu ? 'block' : 'none';
    }
}
    } else if (e.ids[i] === 'formfield properties' &&
pdfViewer.selectedItems.formFields.length !== 0) {
        search.style.display = 'block';
    }
}
}
}
    public lockAnnotations(e: any) {
        var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
        for (var i = 0; i < pdfViewer.annotationCollection.length; i++) {
            if (pdfViewer.annotationCollection[i].uniqueKey ===
pdfViewer.selectedItems.annotations[0].id) {
                pdfViewer.annotationCollection[i].annotationSettings.isLock = true;
                pdfViewer.annotationCollection[i].isCommentLock = true;

pdfViewer.annotation.editAnnotation(pdfViewer.annotationCollection[i]);
            }
            e.cancel = false;
        }
    }
    public unlockAnnotations(e: any) {
        var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
        for (var i = 0; i < pdfViewer.annotationCollection.length; i++) {
            if (pdfViewer.annotationCollection[i].uniqueKey ===
pdfViewer.selectedItems.annotations[0].id) {
                pdfViewer.annotationCollection[i].annotationSettings.isLock = false;
                pdfViewer.annotationCollection[i].isCommentLock = false;

pdfViewer.annotation.editAnnotation(pdfViewer.annotationCollection[i]);
            }
            e.cancel = false;
        }
    }
    public setReadOnlyTrue(e: any) {
        var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
        var selectedFormFields = pdfViewer.selectedItems.formFields;
        for (var i = 0; i < selectedFormFields.length; i++) {
            var selectedFormField = selectedFormFields[i];
            if (selectedFormField) {
                pdfViewer.formDesignerModule.updateFormField(selectedFormField, {
                    isReadOnly: true,
                } as any);
            }
            e.cancel = false;
        }
    }
    public setReadOnlyFalse(e: any) {

```

```

var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
var selectedFormFields = pdfViewer.selectedItems.formFields;
for (var i = 0; i < selectedFormFields.length; i++) {
    var selectedFormField = selectedFormFields[i];
    if (selectedFormField) {
        pdfViewer.formDesignerModule.updateFormField(selectedFormField, {
            isReadOnly: false,
        } as any);
    }
    e.cancel = false;
}
}
public contextmenuHelper(e: any): void {
    var pdfViewer =
(<any>document.getElementById('pdfViewer')).ej2_instances[0];
    this.hide = this.hideObj
    this.toolbar = this.toolbarObj
    pdfViewer.addCustomMenu(this.menuItems, this.hide.checked,
this.toolbar.checked);
}
}

```

APP.MODULE.TS

```

import { NgModule, ViewChild } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { CheckBoxModule } from '@syncfusion/ej2-angular-buttons';
import { PdfViewerModule, LinkAnnotationService, BookmarkViewService,
MagnificationService, ThumbnailViewService,
ToolbarService, NavigationService, TextSearchService, TextSelectionService,
PrintService, AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService } from '@syncfusion/ej2-angular-pdfviewer';
import { AppComponent } from './app.component';
@NgModule({
    declarations: [
        AppComponent
    ],
    imports: [
        BrowserModule,
        PdfViewerModule,
        CheckBoxModule
    ],
    providers: [LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService, AnnotationService,
FormDesignerService, FormFieldsService, PageOrganizerService],
    bootstrap: [AppComponent]
})
export class AppModule { }

```

MAIN.TS

```

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { enableProdMode } from '@angular/core';

```

```
import { AppModule } from './app.module';
import 'zone.js';
enableProdMode();
platformBrowserDynamic().bootstrapModule(AppModule);
```

Note: To set up the **server-backed PDF Viewer**,

Add the below `serviceUrl` in the `app.component.ts` file

```
public service: string = 'https://services.syncfusion.com/angular/production/api/pdfviewer';
```

Within the template, configure the PDF Viewer by adding the `[serviceUrl]='service'` attribute inside the `div` element.

[View sample in GitHub](#)

`PageRenderInitiate` and `PageRenderComplete` event

In Syncfusion PDF Viewer, `pageRenderInitiate` and `pageRenderComplete` actions are events that occur during the rendering process of PDF documents.

pageRenderInitiate

The `pageRenderInitiate` event is triggered when the rendering of a page in the PDF document begins. This event provides developers with an opportunity to perform any necessary initialization or setup before the rendering of the page content commences. It can be utilized to prepare resources, set up rendering parameters, or execute any other actions required before the page rendering process starts.

pageRenderComplete

The `pageRenderComplete` event is triggered when the rendering of a page in the PDF document is completed. This event allows developers to perform cleanup tasks or finalize rendering-related processes after the rendering of the page content finishes. It can be used to release resources, finalize rendering settings, or handle any post-rendering tasks necessary for the application.

``html`

```
<ejs-pdfviewer #pdfViewer id="pdfViewer"
[serviceUrl]='service'
[documentPath]='document'
(pageRenderInitiate)='pageRenderInitiate($event)'
(pageRenderComplete)='pageRenderComplete($event)'
style="height:640px;display:block">
</ejs-pdfviewer>
```

```

``typescript`

```
public pageRenderInitiate(args: any): void {
// This method is called when the page rendering starts
console.log('Rendering of pages started');
```

```

console.log(args)
}
public pageRenderComplete(args: any): void {
// This method is called when the page rendering completes
console.log('Rendering of pages completed');
console.log(args)
}
`

```

The provided code demonstrates how to subscribe to the `pageRenderInitiate` and `pageRenderComplete` events in the Syncfusion PDF Viewer component.

[View sample in GitHub](#)

### Open and Close Bookmark pane programmatically

The PDF Viewer library allows you to open the Bookmark pane programmatically using the **`openBookmarkPane()`** method.

The following steps are used to open the Bookmark.

**Step 1:** Follow the steps provided in the [link](#) to create a simple PDF Viewer sample.

**Step 2:** Insert the following code snippet to implement the functionality for opening the Bookmark pane:

```

`html
<button (click)="openBookmark()">Open Thumbnail Pane</button>
`

`ts
openBookmark() {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
// Open Bookmark pane.
viewer.bookmarkViewModule.openBookmarkPane();
}
`

```

Similarly, to close the Bookmark pane programmatically, employ the following code snippet:

```

`html
<button (click)="closeBookmark()">Close Bookmark Pane</button>
`

`ts
closeBookmark() {

```

```
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];

// Close Bookmark pane.

viewer.bookmarkViewModule.closeBookmarkPane();

}
```

### [View sample in GitHub](#)

#### Locking Form Fields in a PDF document

The PDF Viewer component offers the ability to enable or disable the locking option for form fields within a PDF document. When a form field is locked, it's prevented from being moved, resized, or removed.

#### *Locking Form Fields Programmatically*

Form fields can be locked either by default settings or by utilizing an event and the `isReadOnly` API.

The provided code snippet exemplifies how to lock form fields in a PDF document during the document loading process. It utilizes the `documentLoad` event to add form fields and subsequently sets them as read-only using the `isReadOnly` API. This ensures that users cannot modify the form fields, thereby preserving data integrity.

#### **STANDALONE**

```
import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService, PageOrganizerService,
TextFieldSettings, RadioButtonFieldSettings, InitialFieldSettings,
CheckBoxFieldSettings, SignatureFieldSettings, LoadEventArgs } from
'@syncfusion/ej2-angular-pdfviewer';
@Component({
 selector: 'app-root',
 // specifies the template string for the PDF Viewer component
 template: `<div class="content-wrapper">
<ejs-pdfviewer id="pdfViewer"
[documentPath]='document'
[resourceUrl]='resource'
(documentLoad)="documentLoaded($event)"
style="height:640px;display:block">
</ejs-pdfviewer>
</div>`,
 providers: [LinkAnnotationService, BookmarkViewService,
MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
TextSearchService, TextSelectionService, PrintService,
AnnotationService, FormDesignerService, FormFieldsService,
PageOrganizerService]
})
export class AppComponent implements OnInit {
 public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
designer.pdf';
 public resource: string = "https://cdn.syncfusion.com/ej2/23.1.43/dist/ej2-
pdfviewer-lib";
```



```

ngOnInit(): void {
}
public documentLoaded(e: LoadEventArgs): void {
var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
if (e.documentName === 'form-designer.pdf') {
viewer.formDesignerModule.addFormField('Textbox', {
name: 'First Name',
bounds: { X: 146, Y: 229, Width: 150, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'Middle Name',
bounds: { X: 338, Y: 229, Width: 150, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'Last Name',
bounds: { X: 530, Y: 229, Width: 150, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('RadioButton', {
bounds: { X: 148, Y: 289, Width: 18, Height: 18 },
name: 'Gender',
isSelected: false,
} as RadioButtonFieldSettings);
viewer.formDesignerModule.addFormField('RadioButton', {
bounds: { X: 292, Y: 289, Width: 18, Height: 18 },
name: 'Gender',
isSelected: false,
} as RadioButtonFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOB Month',
bounds: { X: 146, Y: 320, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOB Date',
bounds: { X: 193, Y: 320, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOB Year',
bounds: { X: 242, Y: 320, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('InitialField', {
name: 'Agree',
bounds: { X: 148, Y: 408, Width: 200, Height: 43 },
} as InitialFieldSettings);
viewer.formDesignerModule.addFormField('InitialField', {
name: 'Do Not Agree',
bounds: { X: 148, Y: 466, Width: 200, Height: 43 },
} as InitialFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Text Message',
bounds: { X: 56, Y: 664, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Email',
bounds: { X: 242, Y: 664, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
}

```

```

viewer.formDesignerModule.addFormField('CheckBox', {
 name: 'Appointment Reminder',
 bounds: { X: 56, Y: 740, Width: 20, Height: 20 },
 isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
 name: 'Request for Customerservice',
 bounds: { X: 56, Y: 778, Width: 20, Height: 20 },
 isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
 name: 'Information Billing',
 bounds: { X: 290, Y: 740, Width: 20, Height: 20 },
 isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
 name: 'My Email',
 bounds: { X: 146, Y: 850, Width: 200, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
 name: 'My Phone',
 bounds: { X: 482, Y: 850, Width: 200, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('SignatureField', {
 name: 'Sign',
 bounds: { X: 57, Y: 923, Width: 200, Height: 43 },
} as SignatureFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
 name: 'DOS Month',
 bounds: { X: 386, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
 name: 'DOS Date',
 bounds: { X: 434, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
 name: 'DOS Year',
 bounds: { X: 482, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
}
var formFields = viewer.retrieveFormFields();
formFields.map((formField: any) => {
 var isReadOnly = { isReadOnly: true };
 viewer.formDesignerModule.updateFormField(formField, isReadOnly);
});
}
}

```

## **SERVER-BACKED**

```

import { Component, OnInit } from '@angular/core';
import { LinkAnnotationService, BookmarkViewService, MagnificationService,
ThumbnailViewService, ToolbarService, NavigationService,
AnnotationService, TextSearchService, TextSelectionService,
PrintService, FormDesignerService, FormFieldsService, PageOrganizerService,
TextFieldSettings, RadioButtonFieldSettings, InitialFieldSettings,

```

```

CheckBoxFieldSettings, SignatureFieldSettings, LoadEventArgs } from
 '@syncfusion/ej2-angular-pdfviewer';
@Component({
 selector: 'app-root',
 // specifies the template string for the PDF Viewer component
 template: `<div class="content-wrapper">
 <ejs-pdfviewer id="pdfViewer"
 [documentPath]='document'
 [serviceUrl]='service'
 (documentLoad)="documentLoaded($event)"
 style="height:640px;display:block">
 </ejs-pdfviewer>
 </div>`,
 providers: [LinkAnnotationService, BookmarkViewService,
 MagnificationService,
 ThumbnailViewService, ToolbarService, NavigationService,
 TextSearchService, TextSelectionService, PrintService,
 AnnotationService, FormDesignerService, FormFieldsService,
 PageOrganizerService]
})
export class AppComponent implements OnInit {
 public document: string = 'https://cdn.syncfusion.com/content/pdf/form-
 designer.pdf';
 public service: string =
 'https://services.syncfusion.com/angular/production/api/pdfviewer';
 ngOnInit(): void {
 }
 public documentLoaded(e: LoadEventArgs): void {
 var viewer = (<any>document.getElementById('pdfViewer')).ej2_instances[0];
 if (e.documentName === 'form-designer.pdf') {
 viewer.formDesignerModule.addFormField('Textbox', {
 name: 'First Name',
 bounds: { X: 146, Y: 229, Width: 150, Height: 24 },
 } as TextFieldSettings);
 viewer.formDesignerModule.addFormField('Textbox', {
 name: 'Middle Name',
 bounds: { X: 338, Y: 229, Width: 150, Height: 24 },
 } as TextFieldSettings);
 viewer.formDesignerModule.addFormField('Textbox', {
 name: 'Last Name',
 bounds: { X: 530, Y: 229, Width: 150, Height: 24 },
 } as TextFieldSettings);
 viewer.formDesignerModule.addFormField('RadioButton', {
 bounds: { X: 148, Y: 289, Width: 18, Height: 18 },
 name: 'Gender',
 isSelected: false,
 } as RadioButtonFieldSettings);
 viewer.formDesignerModule.addFormField('RadioButton', {
 bounds: { X: 292, Y: 289, Width: 18, Height: 18 },
 name: 'Gender',
 isSelected: false,
 } as RadioButtonFieldSettings);
 viewer.formDesignerModule.addFormField('Textbox', {
 name: 'DOB Month',
 bounds: { X: 146, Y: 320, Width: 35, Height: 24 },
 } as TextFieldSettings);
 viewer.formDesignerModule.addFormField('Textbox', {

```

```

name: 'DOB Date',
bounds: { X: 193, Y: 320, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOB Year',
bounds: { X: 242, Y: 320, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('InitialField', {
name: 'Agree',
bounds: { X: 148, Y: 408, Width: 200, Height: 43 },
} as InitialFieldSettings);
viewer.formDesignerModule.addFormField('InitialField', {
name: 'Do Not Agree',
bounds: { X: 148, Y: 466, Width: 200, Height: 43 },
} as InitialFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Text Message',
bounds: { X: 56, Y: 664, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Email',
bounds: { X: 242, Y: 664, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Appointment Reminder',
bounds: { X: 56, Y: 740, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Request for Customerservice',
bounds: { X: 56, Y: 778, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('CheckBox', {
name: 'Information Billing',
bounds: { X: 290, Y: 740, Width: 20, Height: 20 },
isChecked: false,
} as CheckBoxFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'My Email',
bounds: { X: 146, Y: 850, Width: 200, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'My Phone',
bounds: { X: 482, Y: 850, Width: 200, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('SignatureField', {
name: 'Sign',
bounds: { X: 57, Y: 923, Width: 200, Height: 43 },
} as SignatureFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOS Month',
bounds: { X: 386, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {

```

```

name: 'DOS Date',
bounds: { X: 434, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
viewer.formDesignerModule.addFormField('Textbox', {
name: 'DOS Year',
bounds: { X: 482, Y: 923, Width: 35, Height: 24 },
} as TextFieldSettings);
}
var formFields = viewer.retrieveFormFields();
formFields.map((formField: any) => {
var isReadOnly = { isReadOnly: true };
viewer.formDesignerModule.updateFormField(formField, isReadOnly);
});
}
}

```

## Troubleshooting

Experience the Standalone PDF Viewer Component by copying assets from `node_modules` into my app

It is must to copy the assets from `node_modules` in to your app to experience the new Standalone PDF Viewer component. It offers flexibility across different build systems, remaining both framework-agnostic and independent of bundlers. Even without a bundler, you can seamlessly integrate the PDF Viewer by directly linking its assets into your app.

This strategic approach to lazy loading prevents unwieldy file sizes that a single bundle might impose, which is often impractical.

Assets from 'ej2-pdfviewer-lib' need to be manually incorporated due to their on-demand loading. This necessity arises because the host application lacks inherent awareness of these assets' lazy loading behavior.

### Troubleshoot error 'cp' is not recognized as a command

The error message you're seeing, "'cp' is not recognized as an internal or external command," is because the `cp` command you're trying to use is not recognized on Windows command prompt.

On Windows, you should use the `copy` command to copy files and directories instead of `cp`. The equivalent command in Windows to copy a directory and its contents recursively is:

```
`batch
```

```
xcopy /s /e /i .\node_modules\@syncfusion\ej2-pdfviewer\dist\ej2-pdfviewer-lib src\assets\ej2-
pdfviewer-lib
```

```
`
```

Here, `/s` indicates that you want to copy directories and subdirectories recursively. Also, note that Windows uses backslashes `\` as path separators, not forward slashes `/`.

Make sure to run this command in the appropriate directory where you want to perform the copy operation.

**Note:** If you encounter other issues or error messages while working with the Windows Command Prompt, make sure to double-check your command syntax and file paths for accuracy. Additionally,

ensure that you have the necessary permissions to perform the copy operation in the specified directories.

#### Document Loading Issues in Version 23.1 or Newer

If you're experiencing problems with your document not rendering in the viewer, especially when using version 23.1 or a newer version, follow these troubleshooting steps to resolve the issue:

- **Check for viewer.dataBind() Requirement:** Ensure that you have called `viewer.dataBind()` as required in version 23.1 or newer. This explicit call is essential for initializing data binding and document rendering correctly. It is must to call the `dataBind()` method before load.

```
`typescript
<button (click)="documentLoad()">Load</button>
<ejs-pdfviewer id="pdfViewer"
style="height:640px;display:block">
</ejs-pdfviewer>
function documentLoad () {
var pdfviewer = (<any>document.getElementById("pdfViewer")).ej2_instances[0];
pdfViewer.serviceUrl = "https://ej2services.syncfusion.com/angular/development/api/pdfviewer";
pdfViewer.dataBind();
pdfViewer.load("https://cdn.syncfusion.com/content/pdf/annotations.pdf",null);
}
`
```

- **Verify Document Source:** Confirm that the document source or URL you're trying to display is valid and accessible. Incorrect URLs or document paths can lead to loading issues.
- **Network Connectivity:** Ensure that your application has a stable network connection. Document rendering may fail if the viewer can't fetch the document due to network issues.
- **Console Errors:** Use your browser's developer tools to check for any error messages or warnings in the console. These messages can provide insights into what's causing the document not to load.
- **Loading Sequence:** Make sure that you're calling `viewer.dataBind()` and initiating document loading in the correct sequence. The viewer should be properly initialized before attempting to load a document.
- **Update Viewer:** Ensure that you're using the latest version of the viewer library or framework. Sometimes, issues related to document loading are resolved in newer releases.
- **Cross-Origin Resource Sharing (CORS):** If you're loading documents from a different domain, ensure that CORS headers are correctly configured to allow cross-origin requests.
- **Content Security Policies (CSP):** Check if your application's Content Security Policy allows the loading of external resources, as this can affect document loading. Refer [here](#) to troubleshoot.

By following these troubleshooting steps, you should be able to address issues related to document loading in version 23.1 or newer, ensuring that your documents render correctly in the viewer.

### Uncaught DOMException: Failed to execute 'importScripts' on 'WorkerGlobalScope'

Another error that can occur when setting up the `ej2-pdfviewer-library` without the necessary assets is the **Uncaught DOMException: Failed to execute 'importScripts' on 'WorkerGlobalScope'**. This error typically arises when a web worker attempts to load a script (e.g., `pdfium.js` or `pdfium.wasm`) and fails to do so.

To troubleshoot and resolve this error, consider the following steps:

1. **Check Asset Availability:** Ensure that the required assets, specifically `pdfium.js` and `pdfium.wasm`, are present in the correct locations within your project. Confirm that they are accessible and have been copied correctly.
2. **Network and CORS:** Verify that there are no network-related issues preventing the web worker from fetching the assets. Also, address any CORS-related problems, especially if the assets are hosted on a different origin.

**Note:** If `ej2-pdfviewer-lib` folder is not available in the `'src/assets'`, copy the contents of the `ej2-pdfviewer-lib` folder from `./node_modules/@syncfusion/ej2-pdfviewer/dist` to the `src/assets` directory using the command:

```
`bash
```

```
cp -R ./node_modules/@syncfusion/ej2-pdfviewer/dist/ej2-pdfviewer-lib src/assets/ej2-pdfviewer-lib
```